Centrify Mobile Authentication Services

SDK Implementation Guide

7 November 2013

Centrify Corporation

Legal notice

Contents

Revision history

This table summarizes the changes that have been made to this document with each revision.

Date	Notes
26 March 2013	First draft of introductory material.
03 April 2013	Minor editorial corrections.
09 April 2013	About and Introduction chapters revised in response to editorial review.
12 April 2013	Added API documentation for the Mobile Authorization SDK, SAML Java SDK, and SAML J2EE SDK.
15 April 2013	Removed protected methods from the SAML Java SDK API documentation.
13 May 2013	Minor editorial corrections.
05 June 2013	Revised Introduction. Incorporated technical edit to SAML SaaS API reference chapter.
12 June 2013	Added chapter on debugging and troubleshooting. Added chapter with a walkthrough of SAML Java sample code.
17 June 2013	Editorial and technical corrections.
19 June 2013	Updated API documentation for the generic Java MAS SDK.
26 June 2013	Added API documentation for the iOS MAS SDK.
10 July 2013	Changed name to *Centrify Mobile Authentication Services SDK Implementation Guide* from *Centrify Cloud Programmer's Guide*.
30 August 2013	Added information on writing SAML script for Centrify Cloud Manager.
02 October 2013	Added chapter with walkthrough of SAML J2EE sample code.
23 October 2013	Added chapters with walkthroughs of Android and iOS MAS sample code, Android and iOS integration information, and Android and iOS MAS troubleshooting information.
06 November 2013	Updated Android MAS reference with new exceptions.

About this guide

This document describes the use of three software development kits (SDKs):

- Centrify® Mobile Authentication Service (MAS) Android SDK helps an Android mobile application authenticate a user through Centrify Cloud Service
- Centrify MAS iOS SDK helps an iOS mobile application authenticate a user through Centrify Cloud Service
- Centrify SAML SaaS SDK helps a web (software as a service, or SaaS) application add an authentication interface using Security Assertion Markup Language (SAML) that can be used to authenticate users through Centrify Cloud Service

This document also describes how to write a SAML script to help Centrify Cloud Service generate a SAML authentication token.

Centrify Cloud Service authenticates users through Active Directory. By using the APIs described in this document, you can obviate the need to request or manage user names or passwords for users of your SaaS and mobile applications.

Intended audience

This document is intended for developers and network administrators who are responsible for managing SaaS or mobile applications and who want to implement single sign-on (SSO) using Centrify Cloud Service. This document has two primary audiences: Saas application developers who want to add SAML authentication to their application; and mobile application developers who want to interact with Centrify Cloud Service to implement SSO authentication.

Note that, depending on your needs, you can use the SAML and MAS APIs together or you can use one of the APIs independently of the others. For example, you can add SAML capabilities to a SaaS application that is accessed through a web browser, in which case you do not need the MAS APIs. Or, you can create a mobile application that interfaces with a SaaS application that is already SAML-enabled, and use the MAS API to implement SSO with no need for the SAML SaaS API. On the other hand, you can create a mobile application that uses Centrify Cloud Service to provide SSO to a SaaS application that is not already SAML-enabled, in which case you might want to use both the MAS and SAML SaaS APIs.

It's important to note that the developer of any mobile application that uses a SAML-enabled web service as a back-end must provide a SAML script that is necessary for Centrify Cloud Service to generate a SAML authentication token. The SAML application uses this token (also called a SAML response) to authenticate the user. The last chapter of this book describes how to write a SAML script.

This document assumes you are familiar with programming principles and the syntax of the Java programming language, and can read sample code written in Java.

Using this guide

Chapter 1, "Introduction," describes the purpose and use of the APIs.

Chapter 2, "Integration of Android MAS Apps," gives a step-by-step procedure for integrating Android mobile applications.

Chapter 3, "Integration of iOS MAS Apps," gives a step-by-step procedure for integrating Android mobile applications.

Chapter 4, "Android MAS API Troubleshooting and Debugging," answers common questions encountered when testing and integrating a mobile application that uses the Android MAS API.

Chapter 5, "iOS MAS API Troubleshooting and Debugging," answers common questions encountered when testing and integrating a mobile application that uses the Android MAS API.

Chapter 6, "Mobile Authentication Service Android Reference," documents the classes and methods of the Mobile Authentication Service Android API, for use by Android developers writing a mobile application for use with Centrify Cloud Service.

Chapter 7, "Android MAS Sample Code," walks through the Java code for a sample mobile application that uses the Android MAS API.

Chapter 8, "Mobile Authentication Service iOS Reference," documents the classes and methods of the Mobile Authentication Service iOS API, for use by iOS developers writing a mobile application for use with Centrify Cloud Service.

Chapter 9, "iOS MAS Sample Code," walks through the Objective-C code for a sample mobile application that uses the iOS MAS API

Chapter 10, "SAML Java Reference," documents the classes and methods of the SAML SaaS Java API, a low-level API for use by developers who are adding SAML capabilities to a SaaS application for use with Centrify Cloud Service.

Chapter 11, "SAML Java Sample Code," walks through the Java code for web-server-based routines that read and parse SAML responses using the SAML Java API.

Chapter 12, "SAML J2EE Reference," documents the classes and methods of the SAML SaaS J2EE API, a high-level API for use by developers who are adding SAML capabilities to a SaaS application for use with Centrify Cloud Service.

Chapter 13, "SAML J2EE Sample Code," walks through the Java code for routines that implement a filter and a login servlet using the SAML J2EE API.

Chapter 14, "Creating a SAML application profile," describes the SAML application profile needed to enable Centrify Cloud Services to generate a SAML response, explains the steps needed to put SAML profiles in place, and documents the methods, variables, and functions

needed in order to write the SAML script that is an integral part of each SAML application profile.

Conventions used in this guide

The following conventions are used in this guide:

- `Fixed-width font` is used for sample code, program names or output, file names, and commands that you type at the command line. When *`italicized`*, the fixed-width font is used to indicate variables.
- *Italics* are used for book titles.
- **Boldface font** is used for User Interface text and for emphasis.

Where to go for more information

The documentation set for Centrify Cloud Service (that is, Centrify for Mobile and Centrify for SaaS) includes:

- *Release Notes* included on the distribution media or in the download package provide the most up-to-date information about the current release, including system requirements and supported platforms, and any additional information, specific to this release, that may not be included in other documentation.
- *Centrify Cloud Management Suite Installation and Configuration Guide* provides information related to installing the Centrify Cloud Management Suite, which includes the Centrify Cloud Proxy Server and other components. This guide also provides details for configuring the Centrify Cloud Proxy Server. To get the *Installation and Configuration Guide*, go to http://www.centrify.com/cloud/download.asp#documentation.
- *Cloud Manager online help* provides task-oriented information for administrators who need to modify applications, manage roles and users, and configure settings in Cloud Manager. To open this help, click Help from the user name menu in Cloud Manager.
- *Cloud Manager Application Configuration help* provides specific details for configuring each kind of application that Centrify provides—individual SaaS applications for SSO, user-password applications, and mobile applications. To open this help, click the Help link from an application in the App Catalog or an Application Settings dialog box.
- *Centrify help* provides task-oriented information for users to navigate and launch their deployed applications, view their activity, manage their own mobile devices, and specify some Active Directory settings. To open this help, click Help from the user name menu in the Centrify user portal.

To learn more about SAML, you can start with the following documents:

- Jeff Hodges, Neustar, *How to Study and Learn SAML*, 2006 (http://identitymeme.org/doc/draft-hodges-learning-saml-00.html) defines SAML terms and concepts and provides a reading list.
- J. Hughes, E. Maler, et al, OASIS, *Security Assertion Markup Language (SAML) V2.0 Technical Overview*, 2005 (https://www.oasis-open.org/committees/download.php/14361/sstc-saml-tech-overview-2.0-draft-08.pdf) provides an introduction to some of the SAML assertions (requests and responses).

In addition, you can find the answers to common questions, ask new questions, or get best practice guidance by visiting the Centrify Express for Mobile community site (http://community.centrify.com/).

Contacting Centrify Corporation

If you have questions or comments, we look forward to hearing from you. For information about contacting Centrify Corporation, visit our Web site at **www.centrify.com**. From the Web site, you can get the latest news and information about product support and services, upcoming events, investor relations, and sales. In addition, you can find our office locations and a list of our current partners.

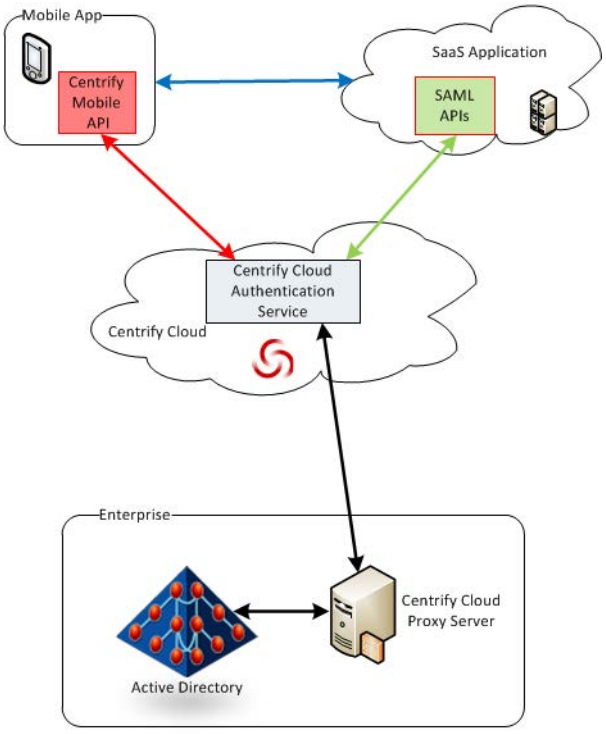
For information about purchasing or evaluating Centrify products, send email to info@centrify.com.

Introduction

As illustrated in Figure 1. Centrify cloud components, there are four main components involved in Centrify Cloud Service:

- A corporate or organization's network running the Centrify Cloud Proxy Server and Active Directory.
- Centrify cloud authentication service.
- One or more providers of Internet (that is, cloud) based software (software as a service, or SaaS) including web applications that may be on-premises or hosted remotely.
- Mobile or desktop devices running software that needs to access the web applications, where the users of the devices belong to groups and roles set up in Active Directory.

Figure 1. Centrify cloud components



Centrify Cloud Service provides authentication and single sign-on (SSO) services for mobile and SaaS applications.

This chapter covers the following topics:

- Centrify SSO enhances the user's experience and improves security

Centrify SSO enhances the user's experience and improves security

Employees of a large organization typically have to use several web-based and mobile applications on a daily basis, such as file-sharing applications, business-networking applications, email, and so forth. Whether they access the applications through a web browser or through applications on a mobile device, they normally have to provide a username and password for each application, each time they open the application. Because it's difficult to remember a large number of passwords, users often use one or a small set of passwords for all the applications they need to use. If one of those applications has a security vulnerability and a user's password is compromised, a malicious hacker might gain access to all of the employee's data in all of the applications that employee uses.

Centrify Cloud Service provides authentication using industry-standard certificate-based security and the organization's Active Directory service. For SaaS applications, the organization provides users with the Centrify portal through which they can access all the applications with which the organization has an account. The user logs in once with his or her Active Directory credentials to gain access to all the SaaS applications that use Security Assertion Markup Language (SAML) protocol to authenticate users. For mobile applications that use Centrify Mobile Authentication Service, a similar situation applies: once the user has enrolled the mobile device in Centrify Cloud Service, he or she has access to every SSO application on the device. Most such mobile applications have a web service providing data in the background, and the SSO extends to the web service without any additional credentials from the user.

MAS enables mobile applications to authenticate users

Mobile applications can use the Centrify Mobile Authentication Service (MAS) APIs to authenticate with Centrify Cloud Service.

On an Android or iOS device, the user installs the Centrify application to obtain access to Centrify Cloud Service. The Centrify application establishes a certificate-based trust relationship with Centrify Cloud Service. The first time the user opens the Centrify application or any other application on the device uses the MAS API to authenticate the user, Centrify Cloud Service prompts the user for Active Directory credentials,

authenticates the user of the device, validates that the user has a current Active Directory account, and looks up the user's roles to determine which applications the user is allowed to run. After that, whenever an application calls the MAS API, the Centrify application provides the user's authentication information. The Centrify application uses the trust relationship it has established with Centrify Cloud Service to get Active Directory and authentication information without any further interaction with the user. The user experience can be described as "zero sign-on" since, after initial enrollment, the user is never required to enter credentials.

Centrify Cloud Service and the Centrify MAS APIs use SAML to authenticate users. The authentication token returned by Centrify Cloud Service to the application is a SAML response. However, because the MAS APIs take care of validating and parsing the token, you do not need to know anything about SAML to implement SSO using the MAS APIs.

The MAS APIs provide general information about the user read by Centrify Cloud Service from the user's Active Directory entry, such as the login user name, the user's address and telephone number, and their email address. If you need other attributes from Active Directory, you can add them to the authentication token using an application-specific SAML script run by Centrify Cloud Service when the user first opens the mobile application. The script is created by the administrator when configuring the Generic SAML template in Centrify Cloud Manager for the application. See "Creating a SAML application profile" on page 225 for instructions on using the Generic SAML template.

A MAS-SDK mobile app authenticates through the Centrify application

When the login sequence is initiated by the user opening a mobile application, the sequence of events is as shown in SSO for an Android or iOS mobile application.

When the user installs and starts up a mobile application that uses the Centrify MAS API, Centrify Cloud Service requests Active Directory credentials from the user and uses them to authenticate and authorize the user. After that the device is recognized by Centrify Cloud Service.

1. The user opens the mobile application and the application requests authentication information using the MAS API.
2. This information might be cached, but if the cached information has expired, the Centrify application contacts Centrify Cloud Service to obtain it using the certificate-based trust relationship set up earlier. In either case, the process is invisible to the user.

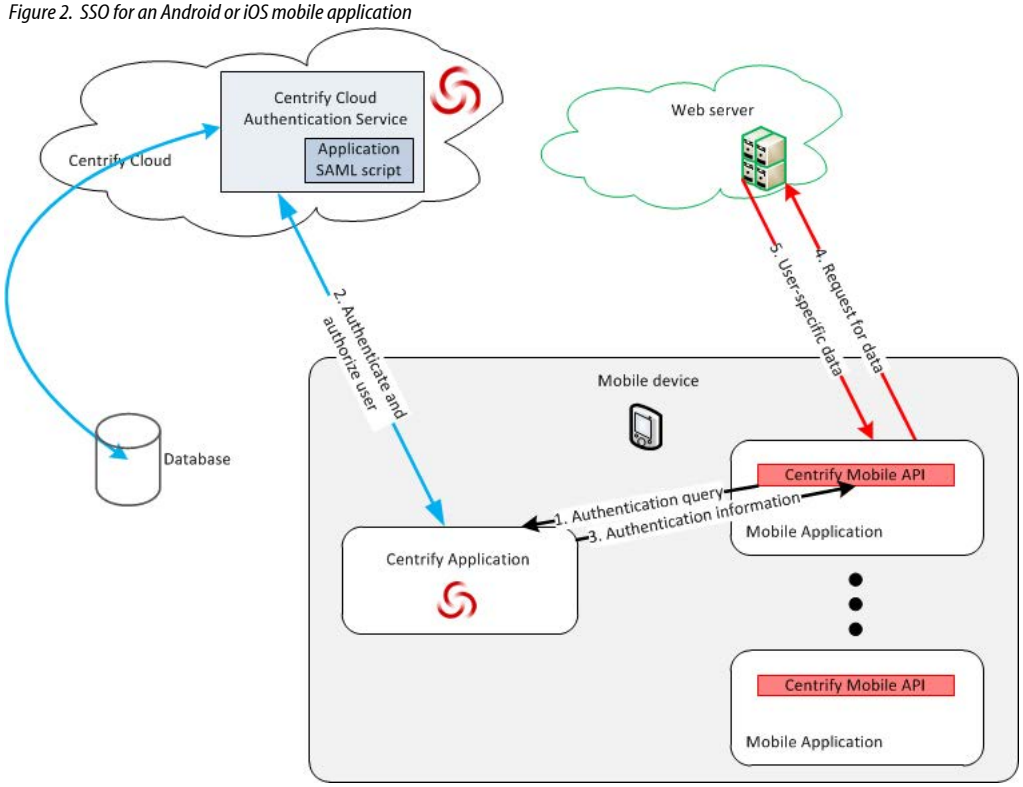
   Note In order for Centrify Cloud Service to generate an authentication token, an administrator must use Centrify Cloud Manager to create and configure an application-specific SAML application profile and script. See "Creating a SAML application profile" on page 225 for instructions on configuring a SAML application profile.
3. The Centrify application returns the information, including the authentication token if requested, to the application.

4 The application passes the authentication token to its web service and requests resources for the user.

5 The application's web service uses the token to authenticate the user and returns the user's data for the application to display.

The user is now logged in and can use the mobile application.

Figure 2. SSO for an Android or iOS mobile application



Provide a SAML application profile to enable authentication tokens

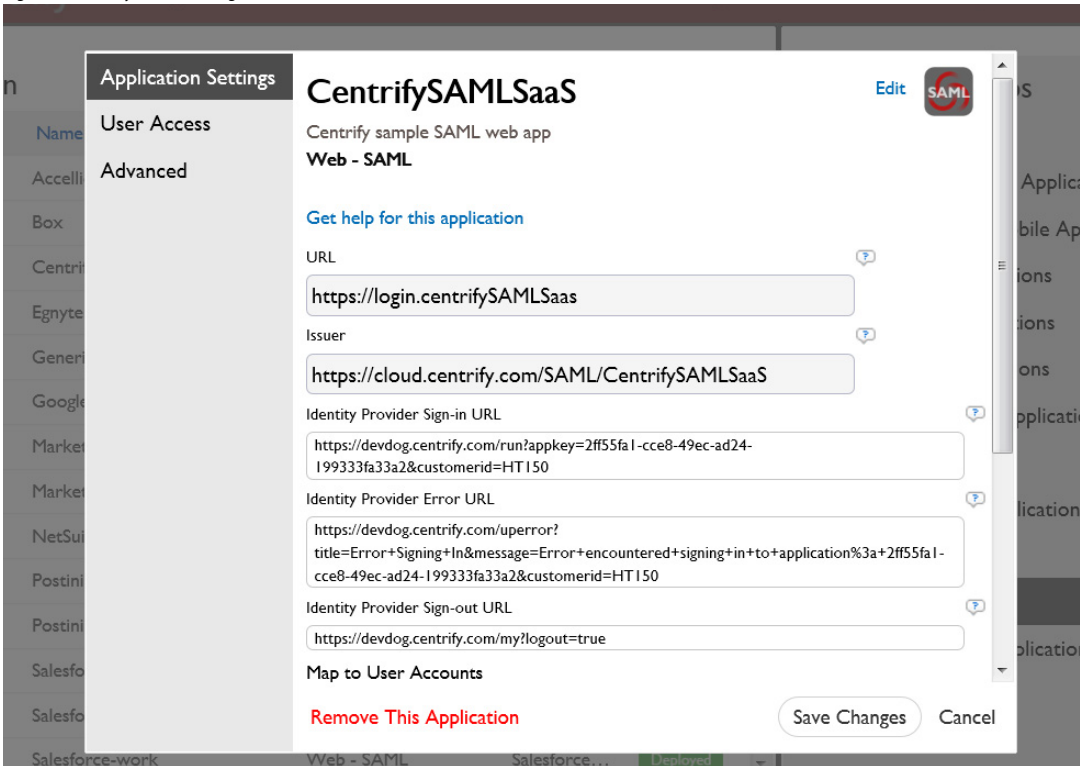
An organization's network administrator uses Centrify Cloud Manager to configure the mobile and SaaS applications that the organization has licensed. The developer who is writing the mobile application that requires an authentication token, the SaaS application, or the web service defines what SAML information the application or service requires in the SAML response (token) presented to the application or service. The organization's administrator configures a SAML application profile in Centrify Cloud Manager for the application or service by using a generic SAML application template to create a custom SAML application profile. Whenever a user in the organization opens an application that requires a SAML response, Centrify Cloud Service uses the SAML application profile to create the response and sends it to the service. For example, your web service might

require the user's telephone number attribute from their Active Directory account. The custom SAML script adds this attribute to the SAML response sent by Centrify Cloud Service.

Note that this need for application-specific information in the SAML interface means that some coordination and exchange of information is needed between the administrator who is adding the SaaS application to Centrify Cloud Manager and the developer or web service provider who is writing or maintaining the SAML interface for the application or web service.

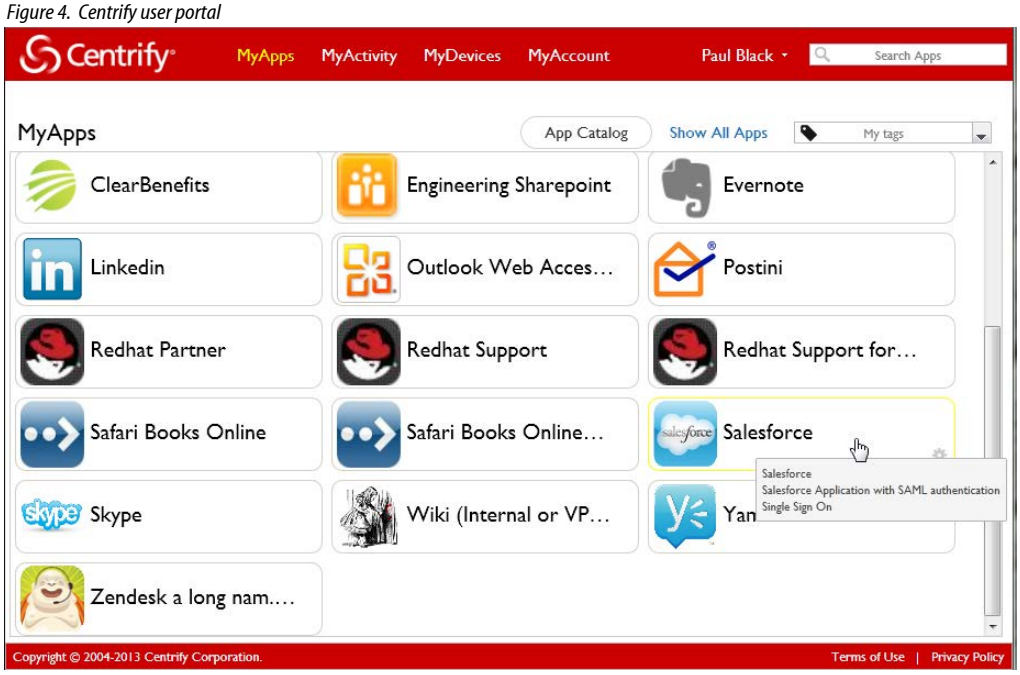
Figure 3. Centrify Cloud Manager shows some of the input fields in Centrify Cloud Manager for the generic SAML template.

Figure 3. Centrify Cloud Manager



When a user logs onto the Centrify user portal, the user gains single sign-on access to the applications configured by the administrator. Figure 4. Centrify user portal shows the Centrify user portal. The Centrify mobile application has a similar list of applications to choose from.

Figure 4. Centrify user portal



SAML assertions are included in the authentication tokens

The authentication token that Centrify Cloud Service returns in response to a call to `EnterpriseAuthentication.getSecurityToken` (in the Android MAS API), `getSecurityTokenForTarget:alwaysUseFreshToken:completionHandler:` (iOS), or `CentrifySaasService.getSSOToken` method (SAML SaaS API) is a SAML response. Depending on your needs, your application can parse the response or pass it on (to an associated web application, for example).

Note The message that carries the authentication information is referred to in SAML protocol terminology as a **response**, although an authentication message of this type need not be in response to a request. Because it can be used to forward authentication information from one location to another, it's sometimes referred to as a **token** or an **authentication token**.

The SAML response contains a SAML assertion, which is the heart of the response. The SAML assertion contains the user's credentials and other information (contained in SAML elements) that the application can read and use for authentication and authorization.

The SAML standard does not define what elements must be present in a SAML assertion. Each application or server that accepts SAML responses defines for itself what elements it requires in an assertion.

The SAML application profile specifies the contents of the token

Whether an application requests a authentication token (SAML response) for itself or in order to pass it on to another application, Centrify Cloud Service must know what information (in the form of SAML elements) to include in the SAML assertion. To create the SAML response, Centrify Cloud Service refers to a stored SAML application profile that defines the SAML assertion elements required for a specific application.

A SAML application profile defines:

- All SAML elements that must be present in the SAML assertion and a value for each of those elements. A piece of Javascript called the SAML script defines all these elements and how values are determined for each element. Centrify Cloud Service executes the SAML script whenever Centrify Cloud Service must create a SAML response and its enclosed SAML assertion. The script may or may not use other properties of the SAML application profile to provide values for the SAML elements in the SAML assertion.
- A URL to which to send the SAML response.
- A URL that identifies Centrify Cloud Service as the SAML identity provider.
- The user name of the user being presented for a SAML-authenticated session.

  Although Centrify Cloud Service requires the user's Active Directory login name for authentication, the user name presented to the web service can be something else. For example, the Threadbare.com web service might use a single user name to authenticate all of Acme Widget's users (`Employee@AcmeWidgets`). When John Doe of Acme Widgets uses the Threadbare mobile application for the first time, Centrify cloud requires him to provide his Active Directory login name (`John.Doe@AcmeWidgets.com`) for authentication purposes, but when Centrify Cloud Service prepares the SAML assertion, it substitutes `Employee@AcmeWidgets` for `John.Doe@AcmeWidgets.com`.

- The security certificate and key used to sign the SAML assertion (or its enclosing SAML response). By default, this is Centrify's certificate and key, but an organization's administrator may specify that the SAML assertion be signed using a different certificate and key—the organization's own certificate and key, for example.
- The identity provider URLs that the SAML service can contact for identity provider sign-in, sign-out, and error reporting. Centrify Cloud Service is the identity provider. These URLs are for direct communication between the SAML service and Centrify Cloud Service; the mobile application or web browser isn't involved. The URLs include unique identifiers for each application profile so Centrify Cloud Service knows what organization the communication concerns.
- A SAML application ID that uniquely identifies this application profile for an organization—"Threadbare Fabrics," for example. The `target` argument for `getSecurityToken` or `getSecurityTokenForTarget:alwaysUseFreshToken:completionHandler:` must

match this name exactly so Centrify Cloud Service knows which SAML application profile to use to create the SAML response.

- A variety of other application profile properties such as an icon and a service description that help administrators work with their organization's application profiles within Centrify Cloud Manager.

The SAML application profile is customized for each organization

A SAML application profile defines many properties that are the same for all organizations using a SAML-enabled service provider. The SAML script, for example, defines the SAML elements and values required by the SAML service, and they don't typically vary from organization to organization. Each organization can, however, customize the SAML assertion to serve their particular needs. The organization may, for example, specify a special user-name mapping or a non-Centrify security certificate.

Because of this, each organization using Centrify Cloud Service for single-sign-on creates its own SAML application profile for each SAML-enabled service provider it uses. To do so, an organization's administrator uses a generic SAML application template in Cloud Manager to create and then configure a new application profile. The administrator can specify in the new profile how to determine a user name within the SAML assertion and what certificate and key to use for the assertion.

The new SAML application profile also presents a unique set of information that the organization gives to the SAML-enabled service provider to define direct communication between the SAML service and Centrify Cloud Service. That information includes the certificate and key used to sign the SAML assertion and the identity provider URLs (sign-on, sign-off, and error).

Each organization must set up a SAML application profile for every SAML-enabled service its users connect to, directly or indirectly. Therefore, when the organization configures Cloud Manager to install an application that uses the Centrify MAS SDK to read or pass on a SAML authentication token, it must provide a SAML application profile customized for the consumer of the authentication token. For example, if the FabricCustomizer mobile application calls `getSecurityToken` in order to pass an authentication token to the SAML interface of the Threadbare Fabrics SaaS application, the administrator for each organization that wants to provide SSO to Threadbare Fabrics through the FabricCustomizer application must not only configure the FabricCustomizer mobile application profile in Centrify Cloud Manager, they must also create and configure a SAML application profile for Threadbare Fabrics. Without a profile in place for a mobile application's associated SAML-enabled web service, an organization's users won't be able to use the mobile application.

For details on SAML application profiles, see Chapter 14, "Creating a SAML application profile."

SAML authentication involves receiving and validating a token

This section describes the steps that occur when a mobile application uses one of the Centrify MAS APIs to request an authentication token and then passes the token to a SAML-enabled web service. This is a typical SAML procedure with the participants taking traditional SAML roles:

- The **principal** is the user of the mobile application, who is authenticated through the Centrify Cloud Service. The mobile application requests an authentication token on behalf of the principal (the user), and passes the token to the SAML-enabled web service, which authorizes the user based on the information in the token.
- The **identity provider** is the Centrify Cloud Service, which authenticates the user through the user's organization's Active Directory and then provides the authentication token (a SAML response) that presents the user as an authenticated principal.
- The **service provider** is the SAML service that receives the SAML response and decides whether to grant resource access to the user.

SAML authentication provides information about the user

When the Centrify Cloud Service authenticates a user, it creates a SAML assertion for the user session that satisfies the requirements of the SAML-enabled web service and includes information about the user that is needed by the web service. Figure 5. SAML authentication process shows the steps Centrify Cloud Service takes when it authenticates a user to a SAML service.

As discussed in "A MAS-SDK mobile app authenticates through the Centrify application" on page 18, the Centrify application on the device provides the SSO service that communicates with Centrify Cloud Service. The mobile application communicates with the Centrify application. Figure 5. SAML authentication process assumes that the user has authenticated with Active Directory and the mobile device has been registered with Centrify Cloud Service as described earlier .

1. The user requests service that requires a response from the SAML service associated with the mobile application.
2. The mobile application calls the MAS API to request an authentication token. The call specifies a SAML application profile to use. If the SSO service (the Centrify application) has cached the token, it returns it directly (skip to Step 8). If not, it sends the request to Centrify Cloud service.
3. Centrify Cloud Service creates a set of Javascript objects for this SAML user session:
   - An **`Application` object** that contains the properties of the SAML service as they're defined in the SAML application profile. Those properties include the application name, the URL, the issuer, the IdP sign-in URL, and others that appear in the

application profile. The `Application` object is a read-only object. A SAML script reads its properties through the object's `get` method.

- A **`LoginUser` object** that contains information about the user identity used to authenticate through the Centrify Cloud Service and the organization's directory service. That information includes the user identity recognized by the SAML service (which is not necessarily the user name supplied through the MAS API call) and so on. This is a read-write object that Centrify Cloud Service sets according to the values on the **Application Settings** tab of the SAML application profile. The SAML script on the **Advanced** tab of the profile uses this object later (Step 5) to set the user name in the SAML assertion.
- A private **assertion object** that defines the elements of the SAML assertion that Centrify Cloud Service builds to send to the web application. This object isn't visible to the custom SAML script, but the SAML script may set the assertion object's properties using a family of global "set" methods (Step 5).

4 Centrify Cloud Service determines the web application log-on user name as it was specified in the SAML application profile, then assigns the name to `LoginUser.Username`.

5 Centrify Cloud Service executes the SAML script to specify a SAML assertion for the user session.

The script must define all the SAML assertion elements required by the web application. The script uses the global assertion-set methods to define the elements in the private assertion object.

6 Centrify Cloud Service creates a SAML assertion based on the properties of the private assertion object and includes the assertion in a SAML response.

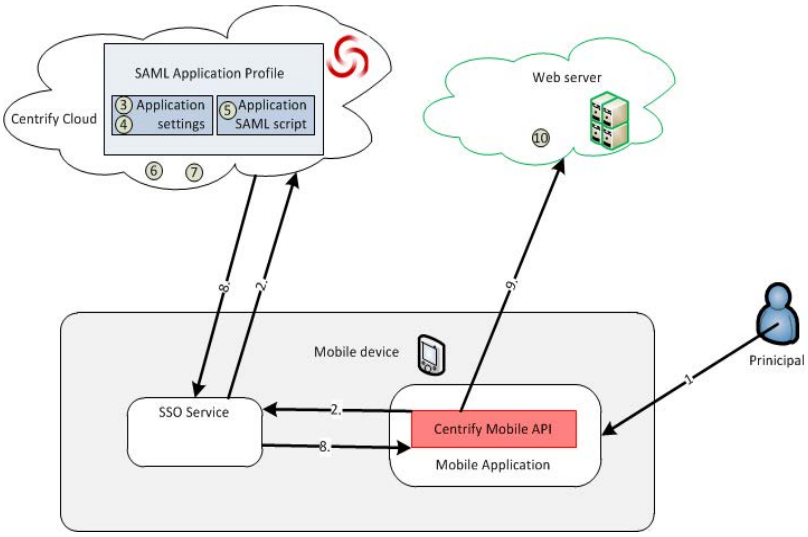
7 Centrify Cloud Service signs the SAML response (or the SAML assertion within the response, depending on what's specified in the custom SAML script). It uses the Centrify certificate private key unless the SAML application profile is set to provide a different certificate.

8 Centrify Cloud Service sends the SAML response to the mobile application.

9 The mobile application sends the SAML response (the authentication token) to the SAML service.

10 The SAML service validates and reads the SAML response and then (if the SAML assertion checks out) logs the user into the SAML service.

Figure 5. SAML authentication process



Use the SAML SaaS API to add SSO to SaaS applications

SaaS applications can use Security Assertion Markup Language (SAML) to authenticate users. You can use the Centrify SAML SaaS API to add this capability to your SaaS application if it doesn't already support SAML.

Although Centrify Cloud Service can securely store a user's username and password and use that data to log in a user to a SaaS application, a true single-sign-on implementation requires SAML. With a username-password SaaS application, the first time the user opens the application through the Centrify user portal (or through a mobile application), the user is required to enter a username and password. Centrify Cloud Service then saves that information so that it can log in the user automatically after that. However, with such an application, even when it appears to be single-sign-on to the user, Centrify Cloud Service has to send the username and password to the SaaS application to log the user on. Therefore, both Centrify Cloud Service and the SaaS application have to store the username and password. If the user should change the password, the single-sign-on would fail.

With SAML, a user logs in once to Active Directory (or unlocks their mobile device) and selects a SaaS application or a mobile application that provides access to a web service. Centrify Cloud Service authenticates the user through Active Directory and returns a SAML response message indicating the user is authenticated. The browser or mobile application forwards the response to the SaaS application, which validates the response and, if it's valid, automatically logs the user in. In this case, neither Centrify Cloud Service nor the SaaS application needs to store the user's password.

Note The message that carries the authentication information is referred to in SAML protocol terminology as a **response**, although an authentication message of this type need

not be in response to a request. Because it can be used to forward authentication information from one location to another, it's sometimes referred to as a **token** or an **authentication token**.

If your SaaS application currently authenticates using usernames and passwords, adding a SAML interface with the Centrify SaaS API means that you shift the authentication burden to Centrify Cloud Service, which uses the users' Active Directory identities to authenticate them. In this case, because it's the users' company or organization that has the license to use your SaaS application, not the individual user, you do not have to authenticate or authorize the user—instead, you can accept a token from Centrify Cloud Service that certifies that the user is authorized to use your application. The token ensures that the user both belongs to an organization that has a valid license and that the user has been authenticated. The token can also contain any additional information you need about the user. Once you've added the SAML interface, your SaaS application does not need to ask for usernames or store passwords for these users. In addition, the users need to log in only once to Active Directory to get access to all the SaaS applications for which their company or organization has a license.

The situation is a bit different for stand-alone mobile applications. The first time the Centrify application or any mobile application that uses the Centrify MAS API calls a method that requires authentication, Centrify Cloud Service presents the user with a dialog requesting Active Directory credentials. Centrify Cloud Service contacts the Centrify Cloud Proxy Server on the network of the organization to which the user belongs, authenticates the user, and grants the user's device zero sign-on (ZSO) privileges. That is, once the user has logged into the device, the user never needs to log in again to any MAS application or to any SAML-enabled web service available through that application.

Note Because any user of the device gains access to the web service without further authentication, it is highly recommended that you use a mobile device management (MDM) solution, such as Centrify Cloud Manager, to require a secure passcode and auto-lock policy for the mobile device.

When a mobile application or SAML-enabled SaaS application requests authentication through a Centrify API, Centrify Cloud Service provides a SAML response signed with a private key. By default, Centrify Cloud Services uses Centrify's signing certificate and private key; however a software vendor can provide a private key and certificate for this purpose. The SAML response includes any information needed by the SaaS application or web service to authorize the user.

The web browser or mobile application passes the response on to the SaaS application or web service, which verifies the signature using the appropriate certificate. If the signature is valid, then the SaaS application or web service can assume that the user has been authenticated and is authorized to use the service. The SaaS application or web service uses the information in the SAML response to log in the user. The user can then use the SaaS application or mobile application in the same way as if he or she had logged on with a username and password.

When using Centrify Cloud Service with SAML authentication, there is never a need for the SaaS application or web service to contact Active Directory directly or to store usernames and passwords.

From the point of view of the SAML-enabled service provider, there are three possible scenarios for user authentication using Centrify Cloud Service:

- Identity-provider-initiated login using the Centrify portal on a computer or mobile device. Centrify Cloud Service is the identity provider (IdP).
- Service-provider-initiated login using a browser on a computer or mobile device. The SaaS application is the service provider (SP).
- Service-provider-initiated login using a mobile application that provides access to the web service.

For each of these scenarios, there are several ways to use the SAML SaaS API:

- Using the high-level J2EE (Java Platform, Enterprise Edition) API, which does most of the work of dealing with SAML responses for you
- Using the lower-level Java API to retrieve the SAML metadata for you and using that to generate SAML requests and validate responses
- Using your own means to obtain the SAML metadata and using the lower-level Java API to generate SAML requests and validate responses

A user can log in to a SaaS application from the Centrify App

The sequence of events when a user initiates the login from the Centrify application is indicated by the numbered arrows in Figure 6. IdP-initiated login. Before starting the sequence shown in the figure, the user logs into the Centrify user portal, or logs into their mobile device and opens the Centrify mobile application. Centrify Cloud Service authenticates the user with the Active Directory server of the user's organization. The authentication is done in the background and indicated with blue arrows in the figure.

1. The user selects a SaaS application in the Centrify user portal.
2. Centrify Cloud Service reads a custom SAML application profile specific to the application that specifies what contents are needed for the SAML response and prepares the response.

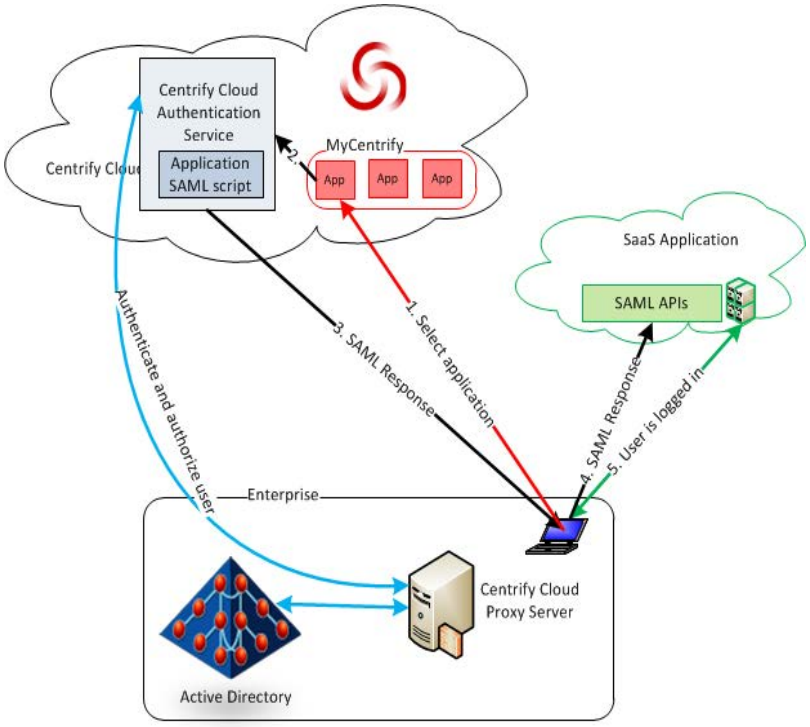
   Note The application-specific SAML script is created by the administrator when configuring the SaaS application in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on configuring a SaaS application.

   Centrify Cloud Service signs the SAML response using the private key specified in Centrify Cloud Manager for that application and sends the response back to the user's browser.

3 The browser sends the SAML response on to the SaaS application, which uses the SAML SaaS API to validate the response and extract the data the SaaS application needs to log in the user.

4 The user is now logged in and can use the SaaS application.

Figure 6. IdP-initiated login



A user can log in to a SaaS application from a browser

When the login sequence is initiated by the user attempting to log in directly to the SaaS application from a browser, the SaaS application (the service provider, or SP) sends a SAML authentication request to the user's browser, which redirects it to Centrify Cloud Service. (The administrator provides the redirection URL when configuring the SaaS application in Centrify Cloud Manager.) From that point on, the sequence of events is identical to that discussed in A user can log in to a SaaS application from the Centrify App. The authentication is done in the background and indicated with blue arrows in the figure.

Note that the authentication request is sent only if the SaaS application is configured to handle this case. If not, the SaaS application can request a username and password or can just refuse to let the user log in.

The sequence of events is shown in SP-initiated login from a browser.

1 The user opens the SaaS application.

2 The SaaS application detects that the user is a Centrify client that has not yet been authorized (for example, the SaaS vendor might provide a unique login URL to a particular client organization) and sends a SAML request (`AuthnRequest`) to the user's browser, addressed to Centrify Cloud Service. (The redirection address is available through the SAML Java and J2EE APIs.)

3 The browser redirects the request to Centrify Cloud Service, which authenticates the user with Active Directory. If the user has not previously been authenticated by Centrify Cloud Service, Centrify Cloud Service displays a dialog requesting the user's Active Directory credentials.

4 Centrify Cloud Service reads a custom SAML application profile specific to the application that specifies what contents are needed for the SAML response and prepares the response.

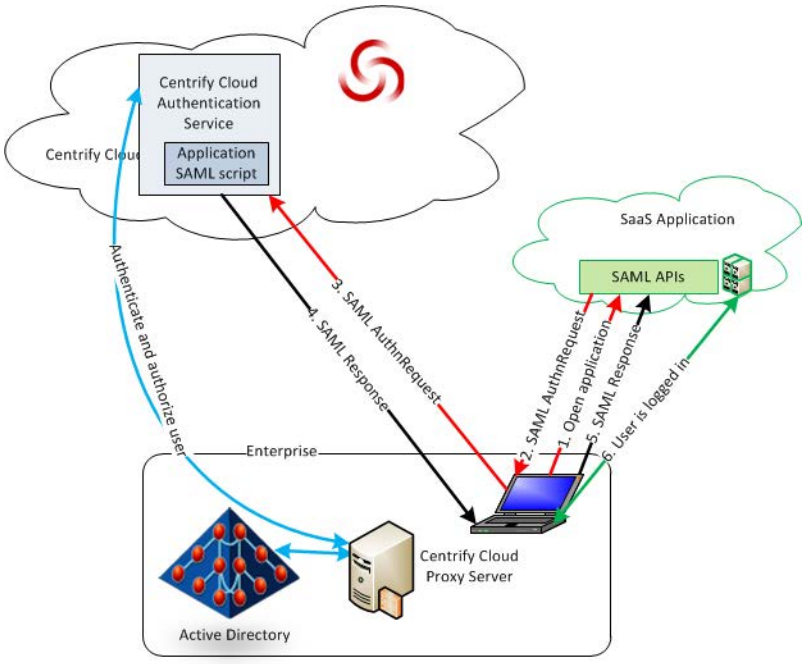
Note The application-specific SAML script is created by the administrator when configuring the SaaS application in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on configuring a SaaS application.

Centrify Cloud Service signs the SAML response using the private key specified in Centrify Cloud Manager for that application and sends a SAML response to the user's browser.

5 The browser sends the response on to the SaaS application, which uses the SAML SaaS API to validate the response and extract the data the SaaS application needs to log in the user.

6 The user is now logged in and can use the SaaS application.

Figure 7. SP-initiated login from a browser



A user can log in to a web service from a mobile application

When the login sequence is initiated by the user attempting to use a MAS-API-enabled mobile application that uses a web service, the sequence is similar to that of the service-provider-initiated login using a browser. The difference is that the mobile application must use the MAS API to issue the SAML request (`AuthnRequest`) to Centrify Cloud Service.

The sequence of events is described in "A MAS-SDK mobile app authenticates through the Centrify application" on page 18.

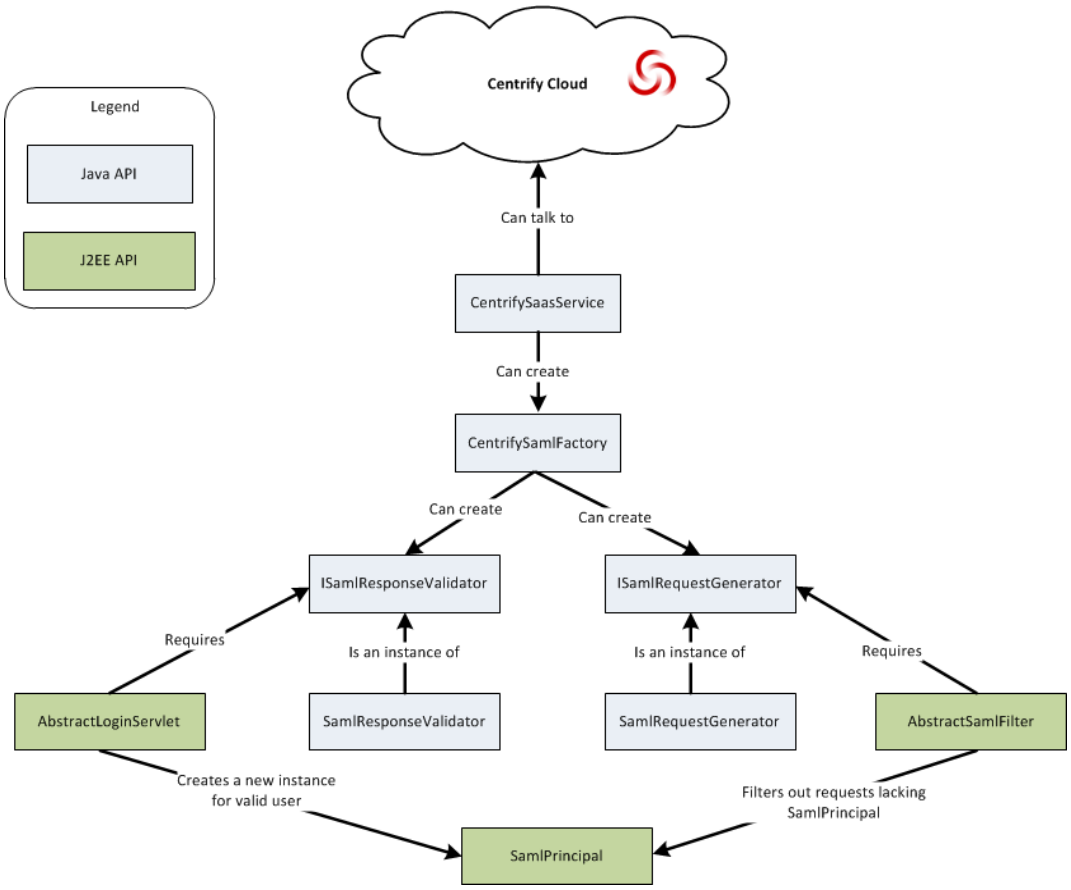
There are Two SAML SaaS APIs

There are two SAML SaaS APIs: a high-level J2EE API and a lower-level Java API.

Note The enterprise Java computing platform from Oracle is known as Java 2 Platform Enterprise Edition (J2EE) for version 1.4 and earlier, and as Java Platform Enterprise Edition (Java EE) for version 5 and later. As the SAML SaaS API supports both J2EE 1.4 and Java EE, it is referred to as J2EE in this document for the sake of simplicity.

The main classes in the APIs and their relationships are shown in SAML SaaS APIs and their main classes.

Figure 8. SAML SaaS APIs and their main classes



The following tables briefly describe the classes. See Chapter 10, "SAML Java Reference" for complete class and method descriptions in the Java API and Chapter 12, "SAML J2EE Reference" for the J2EE API.

The main classes in the Java API are:

Class	Description
`CentrifySaasService`	Connects to Centrify Cloud Service to get the SAML metadata for an application.
`CentrifySamlFactory`	Creates `ISamlResponseValidator` and `ISamlRequestGenerator` from metadata.
`SamlRequestGenerator`	Generates a SAML authentication request.
`SamlResponseValidator`	Validates a SAML authentication response.

The main classes in the J2EE API are:

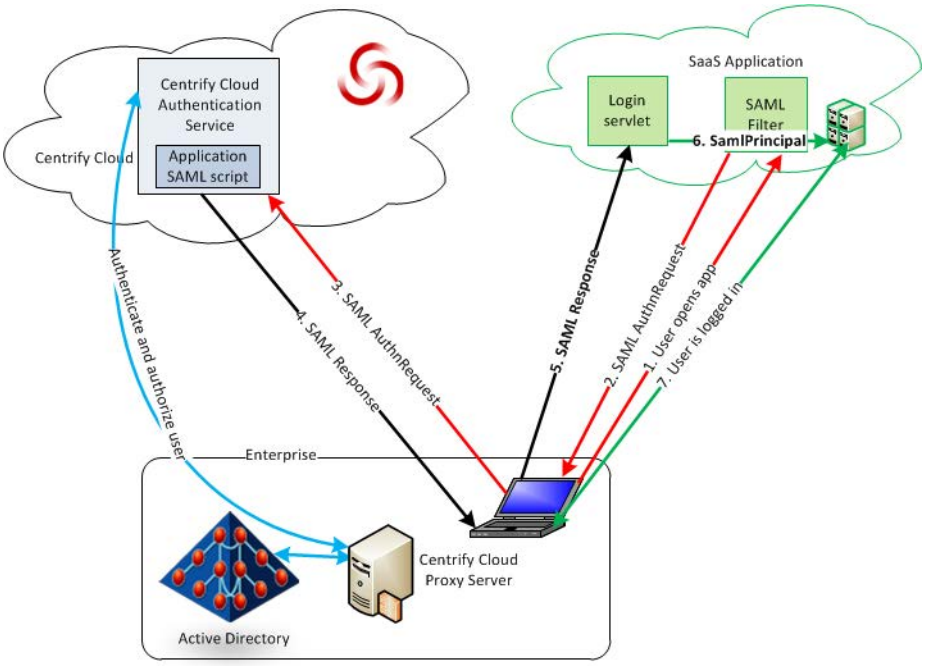
Class	Description
`AbstractLoginServlet`	Implements a code module that receives and processes SAML login requests.
`AbstractSamlFilter`	Implements a code module that checks to make sure a user is logged in before granting access to the SaaS application data.
`SamlPrincipal`	Used by the other two classes as proof that the use is authenticated and authorized.

Using the J2EE SAML SaaS API

The J2EE SAML SaaS API supports J2EE 1.4 or Java EE 5 or later.

The J2EE SAML SaaS API provides templates for the major code modules you need for a SAML interface for a SaaS application. The J2EE API and SP-initiated login from a computer shows how the three main classes of the J2EE SAML SaaS API interact with each other and the rest of the Centrify cloud system in the case of a service-provider-initiated login.

Figure 9. The J2EE API and SP-initiated login from a computer



The general sequence of events is the same as shown in "A user can log in to a SaaS application from a browser" on page 29, however this figure shows more detail for the SaaS application. Specifically:

1. The initial attempt to use the application is intercepted by the filter module (an implementation of the `AbstractSamlFilter` class), which checks to see if the user is authenticated.
2. If not, it generates a SAML authentication request (`AuthnRequest`) to the user's browser, addressed to Centrify Cloud Service. (The redirection address is available through the SAML Java and J2EE APIs).
3. The browser sends the request to Centrify Cloud Service, which authenticates the user.
4. Centrify Cloud Service sends a SAML response to the browser.
5. The browser sends the SAML response back to the SaaS application, where the login servlet (an implementation of the `AbstractLoginServlet` class) processes and validates the response.
6. If the SAML response is valid, the login servlet instantiates a `SamlPrincipal` object and sends it to the SAML filter. The filter now recognizes that the user has been authenticated and passes the login request to the SaaS application.
7. The SaaS application logs in the user.

As illustrated in Figure 8. SAML SaaS APIs and their main classes, the SAML Filter code uses the `SamlRequestGenerator` class to generate the SAML authentication request. The login servlet code uses the `SamlResponseValidator` class to validate the SAML response.

Both the login servlet and the filter can use the `CentrifySamlFactory` class, which can create `ISamlResponseValidator` and `ISamlRequestGenerator` objects. In turn, the `CentrifySamlFactory` class can use the `CentrifySaaSService` class to get SAML metadata from Centrify Cloud Service, which provides much of the data the factory object needs to generate the other objects.

Using the Java API

The Java API works with Java 6 or later.

As shown in SAML SaaS APIs and their main classes and discussed in Using the J2EE SAML SaaS API, the Java API provides a set of lower-level classes that you can use to support the J2EE SAML SaaS API. You can also use the Java API independently if you do not want to use the J2EE runtime environment or want lower-level control of the SAML interface.

The fundamental classes in the Java API are `SamlRequestGenerator` and `SamlResponseValidator`. In fact, you need the `SamlRequestGenerator` class only if you want to support SP-initiated logins. On the other hand, the ability to validate a SAML response is critical to all SAML SSO scenarios.

In order to validate a SAML response, you need the public key corresponding to the private key with which the response was signed. The certificate containing the public key, along with the other parameter values you need to initiate the `ISamlResponseValidator` and `ISamlRequestGenerator` objects, are all available for manual download from the Centrify

Cloud Manager console (Figure 3. Centrify Cloud Manager).

Rather than downloading the certificate and extracting the public key manually, you can download the SAML metadata from the same portal and use the `CentrifySamlFactory` class to generate `ISamlResponseValidator` and `ISamlRequestGenerator` objects.

To avoid having to manually download and save the metadata, you can use the `CentrifySaasService` class, which logs into Centrify Cloud Manager, downloads the metadata, and returns it to you in a format ready to be used as a parameter to the `CentrifySamlFactory` constructor.

You can also use a `CentrifySaasService` object to obtain a SAML response from Centrify Cloud Service or to implement a REST interface (`GET` and `POST` requests) with Centrify Cloud Service. You can use the REST interface to obtain other parameter values needed to instantiate the factory object.

Use the MAS API to add SSO to your mobile application

If your organization uses the Centrify mobile application for iOS or Android—or a browser—to access your web service, and you do not want to write your own mobile application, you do not need the Mobile Authentication Service (MAS) API. However, if you want to write a mobile application that uses Centrify Cloud Service to authenticate users through Active Directory, the Mobile Authentication Service (MAS) API provides some features that make the task much easier.

The MAS API is available from Centrify or from the Appcelerator marketplace at appcelerator.com.

The main classes in the MAS Android API are:

Class	Description
`EnterpriseAuthentication`	Returns authentication information and attributes for a specified user and a SAML response for a specified application.
`SecurityToken`	Gets and sets the information in a SAML response.
`UserAttributes`	Gets a user's Active Directory attributes such as name and address.
`UserInformation`	Gets a user's authentication information.

The main classes in the MAS iOS API are:

Class	Description
`EnterpriseAuthentication`	Returns authentication information and attributes for a specified user and a SAML response for a specified application.
`CentrifySecurityToken`	Gets and sets the information in a SAML response.
`UserAttributes`	Gets a user's Active Directory attributes such as name and address.
`UserInformation`	Gets a user's authentication information.

When your mobile application calls one of the methods of the `EnterpriseAuthentication` class, Centrify Cloud Service checks to make sure the user is authenticated and then returns the requested information. If the user is not authenticated, Centrify Cloud Service prompts the user to enter Active Directory credentials.

You can use the `userAttributes` and `userInformation` objects returned by the `userLookup` and `getUserInformation` methods (Android) or `userLookupForAdditionalAttributes:completionHandler:` and `getUserInformation:` methods (iOS) to check a user's attributes in Active Directory, their user name, and the time they were last authenticated.

To log the user in to a SaaS application, call `EnterpriseAuthentication.getSecurityToken` with the name of the SaaS application. The method returns the SAML response, which you can then post to the SAML interface of the SaaS application. The SaaS application validates the response and, if it's valid, logs in the user.

To examine the SAML response returned to a mobile application, call `EnterpriseAuthentication.getSecurityToken` with the name of the mobile application. Then, pass the SAML response returned by that method to `EnterpriseAuthentication.parseSecurityToken`. That method validates the response and, if the user is authenticated, returns a `SecurityToken` object. You can then use the methods of the `SecurityToken` class to read the contents of the SAML response.

You can customize the SAML response to return information from Active Directory that your application needs. The SAML response contains the attributes that are specified by an application-specific SAML script run by Centrify Cloud Service when the user opens the single-sign-on SaaS or mobile application. The script is created by the administrator when configuring the Generic SAML template in Centrify Cloud Manager for the application. See Chapter 14, "Creating a SAML application profile" for instructions on using the Generic SAML template.

The SAML script specifies the attributes in the token

The custom SAML script specifies elements that must be present in the SAML assertion used to start a session with a SAML service. To write the script, you must know what SAML elements the SAML service requires. The script retrieves the required information from the SAML application profile for the SAML service and from the Active Directory user object for the user who is attempting to log in. The script then specifies the SAML elements and their values. Centrify Cloud Service executes the script while creating a SAML assertion and its enclosing SAML response.

The custom SAML script is written in Javascript. A script is required for each SAML application profile.

To see an example of a SAML script, open the application profile for any SAML application in the **Apps** panel of the Centrify Cloud Manager and click the **Advanced** tab.

Determine the SAML requirements from the vendor of the web application

To write a script for a SAML service, you must determine the requirements of the SAML interface. If you are not the developer of the SAML interface, you must find out from the service's vendor what its SAML requirements are. If the SAML service is a large public application, its vendor may present their SAML requirements on their web site. As an example, Salesforce publishes SAML requirements for authentication at http://login.salesforce.com/help/doc/en/sso_saml_idp_values.htm.

Retrieve information from the Application and LoginUser objects

To retrieve application and user information, use the `Application` and `LoginUser` objects that Centrify Cloud Service creates for a user session.

Retrieving application information

The read-only `Application` object created by the Centrify Cloud Service for a user session contains the properties the administrator enters in the **Application Settings** tab of the SAML application profile.

The method `Application.get` retrieves application properties. It takes as its argument a string that specifies the property whose value to retrieve. `Application.get("Name")`, for example, retrieves the name of the application.

The Application object describes all the Application object's properties that you can retrieve. The following table lists some of the most useful application properties for SAML information. Note that these property names are case-sensitive and that some of the properties have a synonymous global variable that you can use in place of the object property. See "Application" on page 233 for full descriptions.

Property name	Description
Name	The name of the application.
Url	The SAML contact URL.
Issuer	The entity ID, which identifies the Centrify Cloud Service as the IdP.

Retrieving LoginUser properties

The properties of this user session's `LoginUser` object provide information about the user being authenticated for this SAML user session. See "LoginUser" on page 231 for full descriptions of all of the `LoginUser` properties. The following table lists some of the most useful properties for SAML.

Property name	Description
UserName	The user identity presented in the SAML assertion to the SAML service.
GroupNames	Groups in which the user is a direct member.
EffectiveGroupNames	Groups in which the user is an effective member.

Property name	Description
GroupDNs	Distinguished names of groups in which the user is a direct member.
EffectiveGroupDNs	Distinguished names of groups in which the user is an effective member.

Retrieving the user's Active Directory attributes

The `LoginUser` object's `get` method can retrieve any one of the current user's attributes from the organization's Active Directory. It takes as its argument a string that specifies the name of the attribute to retrieve. `LoginUser.get("mail")`, for example, returns the user's email address as stored in Active Directory.

When `LoginUser.get` executes, Centrify Cloud Service contacts Active Directory through the proxy server for the user's organization and retrieves the attribute. If, for example, a user has logged into Centrify as a member of the Threadbare organization, executing `LoginUser.get` during one of that user's log-on sessions contacts the Threadbare Active Directory service through the proxy set up in Threadbare's internal network.

Specify assertion elements in the SAML script

Centrify Cloud Service offers a group of global functions used to set assertion elements for a user session. These functions set the attributes of the private assertion object, which specifies how Centrify Cloud Service constructs the SAML assertion for this user session. Most of these functions take as an argument the value for a specific SAML assertion element. The `setIssuer` function, for example, accepts an entity ID and uses it to specify the issuer URL in the SAML assertion.

Two of these functions, `setAttribute` and `setAttributeArray`, specify a SAML response attribute by name and then specify a value for that attribute that is either a single argument or an array.

The following table lists the most commonly used assertion-set functions. "Global functions" on page 236 fully describes these functions.

global function	Description
setAttribute	Sets a specified SAML assertion element to a value.
setAttributeArray	Sets a specified SAML assertion element to an array.
setAudience	Specifies the audience in an audience restriction in the SAML assertion.
setHttpDestination	Specifies the URL to which to post the SAML response.
setIssuer	Specifies the issuer in the SAML assertion.
setRecipient	Specifies the recipient in the SAML assertion.
setServiceUrl	Specifies the resource to which the user is requesting access.
setSignatureType	Specifies what should be signed: the SAML assertion or the SAML response.

global function	Description
setSubjectName	Specifies the user identity presented to the SAML service.
setVersion	Specifies the version of the SAML assertion.

Use the SAML script template to get started

You can use the SAML script template on the **Advanced** tab of the Generic SAML template in Cloud Manager as a starting point for your own SAML script. The template sets some of the elements described above by retrieving values from the `Application` and `LoginUser` objects. For other elements it provides a sample string such as a URL.

Note that the provided sample string values won't work for your particular SAML service; you need to fill in with values specific to your SAML interface. You may also have to remove some of the sample assertion-set methods if they specify SAML elements that are not required by your SAML service and you may have to add new methods for elements that are required but aren't in the sample script.

Integration of Android MAS Apps

This chapter describes how to use the Mobile Authentication Service (MAS) Android API to create a mobile application designed to run on an Android device. For an application designed to run on an iOS device, see Chapter 3, "Integration of iOS MAS Apps."

Before reading this chapter, you should be familiar with the roles of the MAS API, Centrify Cloud Service, mobile application, and Centrify SSO service application as described in Chapter 1, "Introduction." The MAS Android API is documented in Chapter 6, "Mobile Authentication Service Android Reference."

As described in the Introduction, the MAS API is specifically designed to provide single-sign-on (SSO) capabilities to applications running on an Android device. In this case, the Centrify mobile application authenticates the user through Centrify Cloud Services. The first time the user opens any application that uses the MAS API (including the Centrify application), Centrify Cloud Services prompts the user for an Active Directory user name and password. After that, the user does not need to provide credentials in order to use any of the SSO applications on the device.

Contents of the MAS SDK package

The Android MAS SDK is packaged in the `Android` folder in the `centrify-cloud-SDK` zip file.

See the ReadMe file included in the `Android` folder for an up-to-date enumeration of the files in the SDK. As of the time of this writing, in addition to the ReadMe file, the SDK package contains three folders:

- `centrifyauth2-1.0-SNAPSHOT.jar`: The Centrify MAS SDK library.
- `javadoc`: A folder containing Javadoc documentation for the API. This document (*Centrify Mobile Authentication Services SDK Implementation Guide*) also has documentation for the API; see Chapter 6, "Mobile Authentication Service Android Reference."
- `samples`: A folder containing the following:
  - `GenericSSODemo1.apk`: A demo app.
  - `GenericSSODemo2.apk`: A demo app.
  - `GenericSSODemo1`: A folder containing source code for `GenericSSODemo1.apk`.

Prepare for development

1 Go to https://www.centrify.com/cloud/cloud-service-registration.asp and follow all the steps to register for Centrify Cloud Service and install the Centrify Cloud Proxy Server.

   When you register, Centrify Cloud Service displays a window with a customer ID and proxy activation code. Make a note of the ID and copy the activation code to your clipboard.

2 Connect your production or test Active Directory environment to Centrify Cloud Service as instructed at https://www.centrify.com/cloud/cloud-service-registration.asp.

3 Open Cloud Manager at https://cloud.centrify.com/manage. You need the `sdkdemo` test application in Cloud Manager (necessary for the sample applications to work):

   a Click **Add App**.

   b Search for the `sdkdemo` application in the Select Applications dialog, select it and click **Add Apps**.

   c Follow the directions in the online help to configure the application.

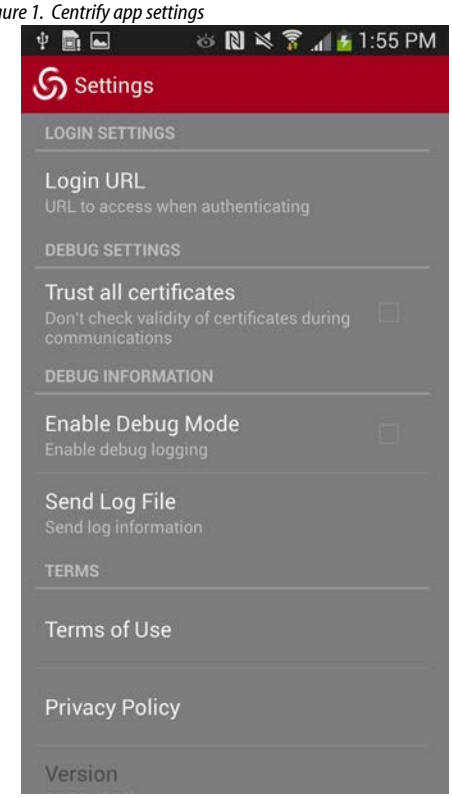
Install and configure the Centrify application

The Centrify application can be run on any mobile device running Android 2.3 or later. The Centrify application must be version 13.10 or later.

1 Download the Centrify application from Google Play and install it on the Android device.

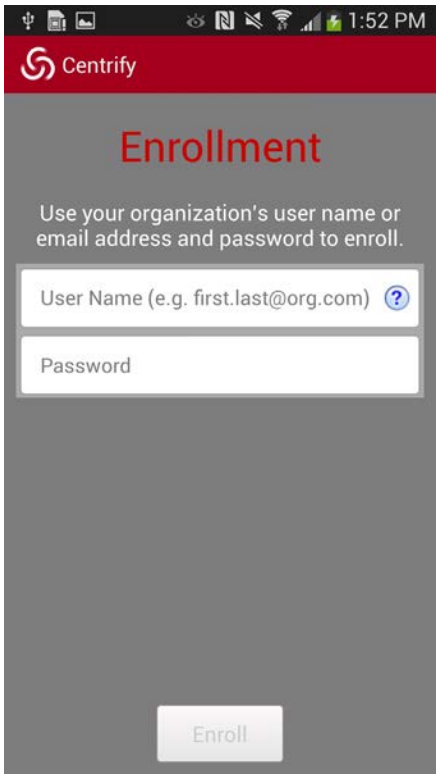
2 Launch the Centrify app and, under **Settings**, make sure the login URL is set to `https://cloud.centrify.com`. Click **Enable Debug Mode**.

Figure 1. Centrify app settings



3 On the Enrollment screen, enter the Active Directory credentials of the user allowed to register the mobile device with Centrify Cloud Service.

Figure 2. Log in with Centrify app



Install and run the demo application

The demo applications can be tested on any mobile device running Android 2.3 or later.

1 Install the `GenericSSODemo1.apk` sample application on the Android device.

Note Android program (apk) files can be installed on both devices and emulators using adb tools, which are part of the Android SDK. For more details, refer to http://developer.android.com/sdk/index.html.

2 Launch the sample application.

3 Try the buttons in the sample application to see the information returned by Centrify Cloud Service.

4 Close the sample application and reopen it. You should not have to enter credentials again.

Create a SAML application profile in Centrify Cloud Manager

You need to add an application profile for your mobile application so that it will appear in the list of available applications in the Centrify application. In addition, you need to add a SAML application profile for your back-end SaaS application so that Centrify Cloud Service can generate the authentication token needed by the SAML interface.

1. Open Centrify Cloud Manager and click **Add App** on the **Apps** page.
2. Enter Android in the search field. Select the Android InHouse template, and click **Add App**.
3. Click `Edit` and enter a name, description, and icon for your mobile application. Click **Save**.
4. Click **Add App** and enter Generic in the search field. Select the Generic SAML template and click **Add App**.
5. Follow the online help instructions and the instructions in Chapter 14, "Creating a SAML application profile" to fill in the template for the back-end web service that works with your mobile application.

Use the sample project to write your application

1. Open the source code for `GenericSSODemo1`.
2. Find the `getSecurityToken` call in the file `SalesRecordsActivity.java`. This method retrieves the authentication token for the application. For example:

```
try {
    String target = Preferences.getTarget(SalesRecordsActivity.this)
    samlToken = EnterpriseAuthentication.getSecurityToken(target, false,
        SalesRecordsActivity.this)
    if (samlToken == null || samlToken.length() == 0) {
        Toast.makeText(SalesRecordsActivity.this,
            "No saml token found", Toast.LENGTH_SHORT).show();
        return null;
    }

    publishProgress("Auth token obtained...");
}
```

   In the sample application, the application name (`target`) passed to the `getSecurityToken` method is `"sdkdemo"`, as defined in the file `Preferences.java`. This call will work only if you have an application named sdkdemo in Centrify Cloud Manager created using the Generic SAML template. Centrify Cloud Services uses the SAML script specified in the Generic SAML template to generate the token.
3. Write your application using the MAS Android API to authenticate the user and parse the authentication token. Be sure that your call to `getSecurityToken` uses the same application name as you used in your Generic SAML template for your web service.

Note You will need to get some parameter values for the methods in the API from Cloud Manager. See the API descriptions in Chapter 6, "Mobile Authentication Service Android Reference" for details.

Prepare your application for use

1. Test your mobile application for SSO.
2. Submit your compiled and tested Android application (`*.apk` file) to Centrify at devsupport@centrify.com.
3. Provide the following information to Centrify so that Centrify can perform quality assurance testing and can include the application in the Centrify Application catalog:
   - Any configuration of the SAML template in Cloud Manager needed to support your SAML interface; for example, the contents of the **URL** and **Issuer URL** fields and the script from the **Advanced** tab.
   - At least one test account credential for the service that you have enabled for SSO.

Contact Centrify for support

If you have any questions, comments, or feedback, please email Centrify at devsupport@centrify.com.

Integration of iOS MAS Apps

This chapter describes how to use the Mobile Authentication Service (MAS) iOS API to create a mobile app designed to run on an iOS device. For an app designed to run on an Android device, see Chapter 2, "Integration of Android MAS Apps."

Before reading this chapter, you should be familiar with the roles of the MAS API, Centrify Cloud Service, mobile app, and Centrify SSO service app as described in Chapter 1, "Introduction." The MAS iOS API is documented in Chapter 8, "Mobile Authentication Service iOS Reference."

As described in the Introduction, the MAS API is specifically designed to provide single-sign-on (SSO) capabilities to apps running on an iOS device. In this case, the Centrify mobile app authenticates the user through Centrify Cloud Services. The first time the user opens any app that uses the MAS API (including the Centrify app), Centrify Cloud Services prompts the user for an Active Directory user name and password. After that, the user does not need to provide credentials in order to use any of the SSO apps on the device.

Contents of the MAS SDK package

The iOS MAS SDK is packaged in the `iOS` folder in the `centrify-cloud-SDK` zip file.

See the ReadMe file included in the `iOS` folder for an up-to-date enumeration of the files in the SDK. As of the time of this writing, in addition to the ReadMe file, the SDK package contains three folders:

- `Libs`: The Centrify MAS framework header files.
- `Doc`: A folder containing documentation for the API. This document also has documentation for the API; see Chapter 8, "Mobile Authentication Service iOS Reference."
- `Samples`: A folder containing source code and the Xcode project for the `CentrifySDKSample` sample app.

Prepare for development

1. Go to https://www.centrify.com/cloud/cloud-service-registration.asp and follow all the steps to register for Centrify Cloud Service and install the Centrify Cloud Proxy Server.

When you register, Centrify Cloud Service displays a window with a customer ID and proxy activation code. Make a note of the ID and copy the activation code to your clipboard.

2 Connect your production or test Active Directory environment to Centrify Cloud Service as instructed at https://www.centrify.com/cloud/cloud-service-registration.asp.

3 Open Cloud Manager at https://cloud.centrify.com/manage. You need the `sdkdemo` test app in Cloud Manager (necessary for the sample app `CentrifySDKSample` to work):

   a Click **Add App**.

   b Search for the `sdkdemo` app in the Select apps dialog, select it and click **Add Apps**.

   c Follow the directions in the online help to configure the app.

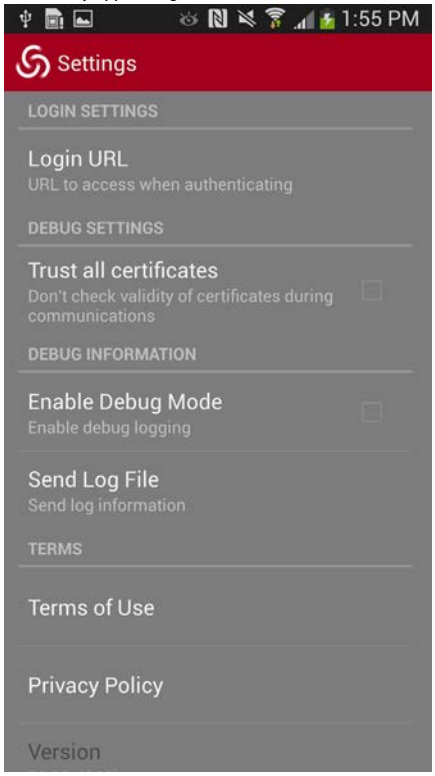
Install and configure the Centrify app

The Centrify app can be run on any mobile device running iOS 6.0 or later. The Centrify application must be version 13.10 or later.

1 Download the Centrify app from the Apple App Store and install it on the iOS device.

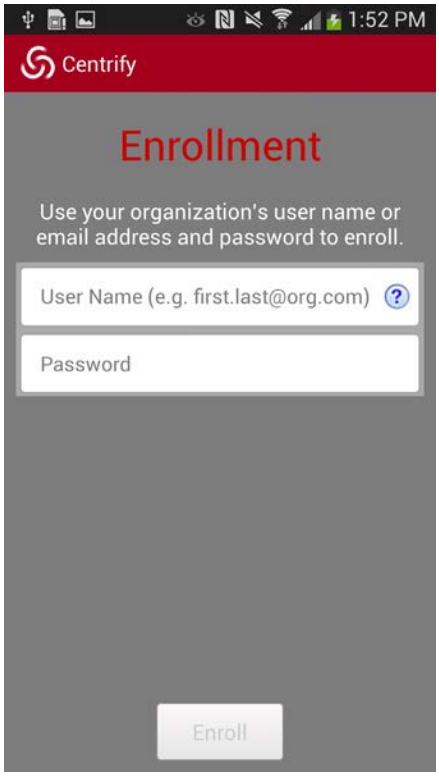
2 Launch the Centrify app and, under **Settings**, make sure the login URL is set to `https://cloud.centrify.com`. Click **Enable Debug Mode**.

Figure 1. Centrify app settings



3 On the Enrollment screen, enter the Active Directory credentials of the user allowed to register the mobile device with Centrify Cloud Service.

Figure 2. Log in with Centrify app



Install and run the demo app

The demo apps can be tested on any mobile device running iOS 6.0 or later.

1 Install the `CentrifySDKSample` sample app on the iOS device or use the emulator in Xcode.

Note For details on how to test an iOS app on a device, refer to https://developer.apple.com/library/mac/documentation/IDEs/Conceptual/AppDistributionGuide/TestingYouriOSApp/TestingYouriOSApp.html.

2 Launch the sample app.

3 Try the buttons in the sample app to see the information returned by Centrify Cloud Service.

4 Close the sample app and reopen it. You should not have to enter credentials again.

Create a SAML app profile in Centrify Cloud Manager

You need to add an app profile for your mobile app so that it will appear in the list of available apps in the Centrify app. In addition, you need to add a SAML app profile for your back-end SaaS app so that Centrify Cloud Service can generate the authentication token needed by the SAML interface.

1. Open Centrify Cloud Manager and click **Add App** on the **Apps** page.
2. Enter iOS in the search field. Select the iOS InHouse template, and click **Add App**.
3. Click `Edit` and enter a name, description, and icon for your mobile app. Click **Save**.
4. Click **Add App** and enter Generic in the search field. Select the Generic SAML template and click **Add App**.
5. Follow the online help instructions and the instructions in Chapter 14, "Creating a SAML application profile" to fill in the template for the back-end web service that works with your mobile app.

Use the sample project to write your app

1. Open the source code for `CentrifySDKSample`.
2. Find the `getSecurityTokenForTarget:` call in the files `AppDelegate.m` and `HomeViewController.m`. This method retrieves the authentication token for the app. For example:

```
- (IBAction)getAccessToken:(id)sender
{
    self.accessToken = nil;
    [EnterpriseAuthentication getSecurityTokenForTarget:@"sdkdemo"
        alwaysUseFreshToken:NO completionHandler:^(CentrifySDKResult *result) {
            [self getSecurityTokenHandler:result];
    }];
}
```

   Notice that, in the sample app, the app name passed to the `getSecurityTokenForTarget:` method is `"sdkdemo"`. This call will work only if you have an app named sdkdemo in Centrify Cloud Manager created using the Generic SAML template. Centrify Cloud Services uses the SAML script specified in the Generic SAML template to generate the token.

3. Write your app using the MAS iOS API to authenticate the user and parse the authentication token. Be sure that your call to `getSecurityTokenForTarget:` uses the same app name as you used in your Generic SAML template for your web service.

   Note You will need to get some parameter values for the methods in the API from Cloud Manager. See the API descriptions in Chapter 8, "Mobile Authentication Service iOS Reference" for details.

Prepare your app for use

1. Test your mobile app for SSO.
2. For a publicly-available app, submit the app to Apple. See https://developer.apple.com/support/iOS/.
3. Submit your compiled and tested iOS app to Centrify at devsupport@centrify.com.
4. Provide the following information to Centrify so that Centrify can perform quality assurance testing and can include the app in the Centrify app catalog:
   - Any configuration of the SAML template in Cloud Manager needed to support your SAML interface; for example, the contents of the **URL** and **Issuer URL** fields and the script from the **Advanced** tab.
   - At least one test account credential for the service that you have enabled for SSO.

Contact Centrify for support

If you have any questions, comments, or feedback, please email Centrify at devsupport@centrify.com.

Chapter 4

Android MAS API Troubleshooting and Debugging

This chapter answers common questions that arise while trying to test or integrate a mobile application written using the Android MAS API.

Q Why did my method call return `SecurityProviderNotFoundException`?

A The method is expecting security information to be provided by the Centrify mobile application. For testing purposes, you must install the Centrify mobile application before running your application.

Q Why can't I enroll my device using the credential I have?

A Make sure the server URL points to the cloud portal; for example, `https://cloud.centrify.com`. This URL is under **Settings** in the enroll dialog, or it can be set in the Centrify application.

Q Why does the `getSecurityToken` method fail with `TargetApplicationNotSupportedException`?

A Make sure that the user to whom the device is registered has been assigned an application with the exact name of the `target` argument in the `getSecurityToken` call. Check the name of the application in Cloud Manager.

Q How can I query an arbitrary Active Directory attribute, such as `userPrincipalName`?

A
```
userLookup(context, new
  String[]{"userPrincipalName"}).getAdditionalAttributes().get(
  "userPrincipalName")
```

Q How can I embed Active Directory user information in the SAML response (authentication token)?

A Find the application in Cloud Manager, click on the application and go to the **Advanced** tab. Use the script to customize the response. See *Cloud Manager online help* for instructions on writing a SAML script to customize the authentication token.

Q How can I read the information in the SAML response?

A Use the `EnterpriseAuthentication.parseSecurityToken` method to validate the SAML response and read the information in it.

Q Why do I keep getting the same token from the `getSecurityToken` method, even though I've updated the token?

A Set the `alwaysUseFreshToken` parameter to `true` in your call to `getSecurityToken`.

Q How can we ensure that our back-end web server trusts the SAML response?

A Your web server validates the token by checking that its signature is valid. By default, SAML responses are signed with Centrify's signing key. To download the certificate you need to validate the signature, go to Cloud Manager, find the page for your application, open the **Application Settings** tab, and click the **Download** button in the **Security Certificate** section.

Q How can I use our enterprise signing key rather than the default key?

A If you want to provide your own certificate and private key, go to Cloud Manager, find the page for your application, open the **Application Settings** tab, and click the **Replace** button in the **Security Certificate** section. Note that you must provide both your signing certificate and the associated private key in a PKCS #12 (`*.p12` or `*.pfx`) file.

Q What do I do after integration is finished and the application has been tested successfully?

A Submit the application binary (`*.apk` file) to Centrify at devsupport@centrify.com. Centrify will contact you if they encounter any problems.

Chapter 5

iOS MAS API Troubleshooting and Debugging

This chapter answers common questions that arise while trying to test or integrate a mobile app written using the iOS MAS API.

Q Why did my method call return `SecurityProviderNotFoundException`?

A The method is expecting security information to be provided by the Centrify mobile app. For testing purposes, you must install the Centrify mobile app before running your app.

Q Why can't I enroll my device using the credential I have?

A Make sure the server URL points to the cloud portal; for example, `https://cloud.centrify.com`. This URL is under **Settings** in the enroll dialog, or it can be set in the Centrify app.

Q Why does the `getSecurityTokenForTarget:` method fail with `TargetappNotSupportedException`?

A Make sure that the user to whom the device is registered has been assigned an app with the exact name of the *`target`* argument in the `getSecurityTokenForTarget:` call. Check the name of the app in Cloud Manager.

Q How can I query an arbitrary Active Directory attribute, such as `userPrincipalName`?

A Here's some sample code:

```
- (IBAction)lookupUserAttribute:(id)sender
{
    NSArray *attributes = [NSArray arrayWithObjects:@"userPrincipalName",
            @"wWWHomePage", nil];
    [EnterpriseAuthentication userLookupForAdditionalAttributes:attributes
            completionHandler:^(CentrifySDKResult *result) {
        [self lookupUserAttributeHandler:result];
    }];
}
...
- (void) lookupUserAttributeHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
    else
```

```
    {
        // User Information received
        UserAttributes *userInfo = (UserAttributes*)result.resultData;
        NSLog(@"User lookup attributes Received : %@", userInfo);

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"User Attributes"
            message:[userInfo description] delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
        [alert show];
    }
}
```

Q How can I embed Active Directory user information in the SAML response (authentication token)?

A Find the app in Cloud Manager, click on the app and go to the **Advanced** tab. Use the script to customize the response. See *Cloud Manager online help* for instructions on writing a SAML script to customize the authentication token.

Q How can I read the information in the SAML response?

A Use the `EnterpriseAuthentication parseSecurityToken:` method to validate the SAML response and read the information in it.

Q Why do I keep getting the same token from the `getSecurityTokenForTarget:` method, even though I've updated the token?

A Set the `alwaysUseFreshToken:` parameter to `true` in your call to `getSecurityTokenForTarget:`.

Q How can we ensure that our back-end web server trusts the SAML response?

A Your web server validates the token by checking that its signature is valid. By default, SAML responses are signed with Centrify's signing key. To download the certificate you need to validate the signature, go to Cloud Manager, find the page for your app, open the **app Settings** tab, and click the **Download** button in the **Security Certificate** section.

Q How can I use our enterprise signing key rather than the default key?

A If you want to provide your own certificate and private key, go to Cloud Manager, find the page for your app, open the **app Settings** tab, and click the **Replace** button in the **Security Certificate** section. Note that you must provide both your signing certificate and the associated private key in a PKCS #12 (`*.p12` or `*.pfx`) file.

Q What do I do after integration is finished and the app has been tested successfully?

A To get the app in the Apple App Store, see https://developer.apple.com/support/iOS. Submit the app to Centrify at devsupport@centrify.com. Centrify will contact you if they encounter any problems.

Chapter 6

Mobile Authentication Service Android Reference

This chapter describes the classes, methods, and properties in the Centrify Mobile Authentication Service Android API. The classes for working with Centrify Mobile Authentication Service are defined in the com.centrify.auth package and consist of:

Class	Description
`EnterpriseAuthentication`	Returns authentication information and attributes for a specified user and a SAML response for a specified application.
`NotAuthenticatedException`	An exception that indicates the current device is not authenticated.
`SecurityProviderNotFoundException`	An exception that indicates there's no service handling single-sign-on requests.
`SecurityProviderNotProvisionedException`	An exception that indicates the security provider is not provisioned.
`SecurityToken`	Gets and sets the information in a SAML response.
`SecurityTokenInvalidException`	An exception that indicates that the SAML response is not valid.
`TargetApplicationNotSupportedException`	An exception that indicates that a specified application is not supported by this installation.
`UserAttributes`	Gets a user's Active Directory attributes such as name and address.
`UserInformation`	Gets a user's authentication information.

Class: EnterpriseAuthentication

Returns authentication information and attributes for a specified user and a SAML response for a specified application.

Syntax

```
class EnterpriseAuthentication
```

Methods

The `EnterpriseAuthentication` class provides the following methods:

Method	Description
`getSecurityToken`	Returns the SAML response of the specified application.
`getUserInformation`	Returns authentication information about the currently authenticated user.
`parseSecurityToken`	Parses the SAML response to create a `SecurityToken` object.
`userLookup`	Returns the attributes of the currently authenticated user.
`equals`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

This class authenticates a user and returns information about the user from Active Directory that you can use for authorization purposes. It also returns a SAML response for a specified application that you can use to authenticate the user or can forward to a SaaS application to prove that the user is authenticated. Once the user is authenticated, the mobile device is registered with Centrify Cloud Service so that further calls to the methods of this class do not require reauthentication.

See Also

NotAuthenticatedException

UserInformation

Field: ACTION_UNREGISTER

Triggers the SDK unregister operation.

Class

`EnterpriseAuthentication`

Syntax

```
static final String ACTION_UNREGISTER;
```

Discussion

Use this field to unregister the device so that the next time this mobile application calls one of the methods in this class, the user will have to authenticate again.

Example

The following code sample illustrates the use of the `ACTION_UNREGISTER` field:

```
View logoutButton = findViewById(R.id.logout_button);
logoutButton.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
        Intent intent = new Intent(
           EnterpriseAuthentication.ACTION_UNREGISTER);
        startActivity(intent);
     }
});
```

Constructor: EnterpriseAuthentication

Creates a new `EnterpriseAuthentication` object.

Syntax

```
EnterpriseAuthentication()
```

Method: getSecurityToken

Returns the SAML response of the specified application.

Class

```
EnterpriseAuthentication
```

Syntax

```
String getSecurityToken(
      String target,
      boolean alwaysUseFreshToken,
      Context context
)
```

Parameters

`target` The name, application ID, or application key of the application for which you want the SAML response. The application name is assigned by the administrator in the generic SAML template and is not guaranteed to be unique. The application ID is assigned by the administrator in the **Application ID** field of the **Application Settings** tab of the Generic SAML template and must be unique within an organization. The application key is assigned by Centrify Cloud Manager and is guaranteed to be universally unique.
You can find the Application key in the Cloud Manager window of the generic SAML

template in the **Identity Provider Sign-in URL** field. For example:

```
https://devdog.centrify.com/run?appkey=2ff55fa1-cce8-49ec-ad24-
199333fa33a2&customerid=HT150
```

alwaysUseFreshToken Set `true` to require the method to download the token from the server even if a token is already cached. Set `false` to use the token that's already been downloaded, if one is available.

context The current Android context.

Return value

The authentication token of the specified application encoded in Base64-encoded XML format.

Errors

The `getSecurityToken` method may throw one of the following exceptions:

`NotAuthenticatedException` The user is not authenticated.

`SecurityProviderNotFoundException` There's no service (such as Samsung KNOX) handling single-sign-on requests.

`SecurityProviderNotProvisionedException` The security provider is not provisioned; for example, the customer ID is not set.

`TargetApplicationNotSupportedException` The specified application is not supported.

Discussion

If the user is not already authenticated, Centrify Cloud Service puts up an authentication dialog requesting the user to enter Active Directory credentials. Because this is a blocking call, if the user is prompted to enter a credential, the call does not return until the user enters the credential.

If your mobile application has a back-end web service that uses SAML authentication, then the target is the name, ID, or key of the SaaS application. If you are setting up a standalone mobile application, then the target is the mobile application.

The method searches first for a match to the application ID, then for a match to the application name, then for a match to the application key. If the method finds duplicate application IDs or application names, the method fails.

Because the token used by Centrify Cloud Service for mobile applications is a SAML response, you must use the Generic SAML template in Centrify Cloud Manager to configure the SAML response. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

When you receive this SAML response, you can use it to authenticate the user, or you can pass it to the SAML interface of the web service, which uses the SAML response to log in the user. In the latter case, the user gets access to the web service.

Example

The following code sample illustrates the use of the `EnterpriseAuthentication.getSecurityToken` method:

```
@Override
protected String doInBackground(Void... params) {
   EnterpriseAuthentication entAuth = new EnterpriseAuthentication();
   try {
        samlToken = entAuth.getSecurityToken("cloudapp", false,
             GetTokenActivity.this);
        if (samlToken == null || samlToken.length() == 0) {
             Toast.makeText(GetTokenActivity.this,
                  "No SAML response found", Toast.LENGTH_SHORT).show();
             return null;
        }

        publishProgress("Auth token obtained");
        ssoRequest();
   } catch (TargetApplicationNotSupportedException e) {
         Log.w(TAG, "Get token failed:" + e);
         publishProgress("Failed to obtain token:" + e);
   } catch (Exception e) {
        Log.w(TAG, "Get token failed:" + e);
         publishProgress("Failed to obtain token:" + e);
   }
   // Authentication failed
   return null;
}
```

See Also

parseSecurityToken

SecurityToken

Method: getUserInformation

Returns authentication information about the currently authenticated user.

Class

`EnterpriseAuthentication`

Syntax

```
UserInformation getUserInformation(
      Context context
)
```

Parameters

`context` The current Android context.

Return value

The currently authenticated user's logon name, the authentication type, and the time in milliseconds of the last authentication.

Errors

The `getUserInformation` method may throw the following exception:

`NotAuthenticatedException` The user is not authenticated.

`SecurityProviderNotFoundException` There's no service (such as Samsung KNOX) handling single-sign-on requests.

`SecurityProviderNotProvisionedException` The security provider is not provisioned; for example, the customer ID is not set.

Discussion

If the user is not already authenticated, Centrify Cloud Service puts up an authentication dialog requesting the user to enter Active Directory credentials. Because this is a blocking call, if the user is prompted to enter a credential, the call does not return until the user enters the credential.

Example

The following code sample illustrates the use of the `EnterpriseAuthentication.getUserInformation` method:

```
@Override
protected UserInformation doInBackground(Void... params) {
    EnterpriseAuthentication entAuth = new EnterpriseAuthentication();
    try {
        return entAuth.getUserInformation(GetUserInfoActivity.this);
    } catch (Exception e) {
        Log.w(TAG, "Authentication failed:" + e);
        error = e.toString();
    }
    // Auth failed
    return null;
}
```

See Also

parseSecurityToken

UserInformation

Method: parseSecurityToken

Parses the SAML response to create a `SecurityToken` object.

Class

`EnterpriseAuthentication`

Syntax

```
SecurityToken parseSecurityToken(
      String token,
      Context context
)
```

Parameters

`token` The SAML response encoded in Base64-encoded XML format.
`context` The current Android context.

Return value

The authentication token of the specified application.

Errors

The `parseSecurityToken` method may throw one of the following exceptions:

`NotAuthenticatedException` The user is not authenticated.

`SecurityProviderNotFoundException` There's no service (such as Samsung KNOX) handling single-sign-on requests.

`SecurityProviderNotProvisionedException` The security provider is not provisioned; for example, the customer ID is not set.

`SecurityTokenInvalidException` The SAML response is not valid; for example, the signature does not pass validity tests.

Discussion

This method validates the digital signature of the SAML response and parses the token into a Java object. When you receive a SAML response, you can use it to authenticate the user, or you can pass it to the SAML interface of the SaaS application, which uses the response as a token to log in the user. In the former case, you can use the methods of the `SecurityToken` class to examine the contents of the response. In the latter case, pass the unparsed string version of the SAML response to the SaaS application.

Example

The following code sample illustrates the use of the `EnterpriseAuthentication.parseSecurityToken` method:

```
@Override
protected String doInBackground(Void... params) {
    EnterpriseAuthentication entAuth = new EnterpriseAuthentication();
    try {
        samlToken = entAuth.getSecurityToken("Box", true, GetTokenActivity.this);
        if (samlToken == null || samlToken.length() == 0) {
            Toast.makeText(GetTokenActivity.this, "No SAML response found",
                Toast.LENGTH_SHORT).show();
            return null;
        }

        token = entAuth.parseSecurityToken(samlToken, GetTokenActivity.this);

        publishProgress("Auth token obtained.");
    } catch (Exception e) {
        Log.w(TAG, "Get token failed:" + e);
        publishProgress("Failed to obtain token:" + e);
    }
    // Auth failed
    return null;
}
```

See Also

getSecurityToken

SecurityToken

Method: userLookup

Returns the attributes of the currently authenticated user.

Class

`EnterpriseAuthentication`

Syntax

```
NotAuthenticatedException userLookup(
      Context context,
      String[] additionalAttributes
)
```

Parameters

`context` The current Android context.

`additionalAttributes` The names of Active Directory attributes in addition to the default Active Directory user attributes, if any are required. Pass `null` if no additional attributes are required.

Return value

The user's attributes, such as their name, address, and groups to which they are assigned.

Errors

The `userLookup` method may throw the following exception:

`NotAuthenticatedException` The user is not authenticated.

`SecurityProviderNotFoundException` There's no service (such as Samsung KNOX) handling single-sign-on requests.

`SecurityProviderNotProvisionedException` The security provider is not provisioned; for example, the customer ID is not set.

Discussion

Because this is a blocking call, if the user is prompted to enter a credential, the call does not return until the user enters the credential.

Example

The following code sample illustrates the use of the `EnterpriseAuthentication.userLookup` method:

```
protected UserAttributes doInBackground(Void... params) {
    EnterpriseAuthentication entAuth = new EnterpriseAuthentication();
```

```
    try {
        String[] attributes = new String[]{"proxyaddresses"};
            return entAuth.userLookup(UserLookupActivity.this, attributes);
    } catch (Exception e) {
        Log.w(TAG, "Authentication failed:" + e);
        error = e.toString();
    }
    // Auth failed
    return null;
}
```

See Also

getUserInformation

NotAuthenticatedException

Class: NotAuthenticatedException

An exception that indicates the current device is not authenticated.

Syntax

```
class NotAuthenticatedException
```

Methods

The `NotAuthenticatedException` class provides the following methods:

Method	Description
`equals`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.
`getMessage`	Inherited from `java.lang.Throwable`.
`getStackTrace`	Inherited from `java.lang.Throwable`.
`initCause`	Inherited from `java.lang.Throwable`.
`printStackTrace`	Inherited from `java.lang.Throwable`.
`setStackTrace`	Inherited from `java.lang.Throwable`.
`toString`	Inherited from `java.lang.Throwable`.

Discussion

This exception might mean that the calling application is not whitelisted or that the user did not enter the correct credential. The message contains additional information about why authentication failed.

Constructor: NotAuthenticatedException

Creates a new `NotAuthenticatedException` object.

Syntax

```
NotAuthenticatedException()
NotAuthenticatedException(
      String message
)
```

Parameters

`message` The string to return with the exception.

Class: SecurityProviderNotFoundException

An exception that indicates there's no service handling single-sign-on requests.

Syntax

```
class SecurityProviderNotFoundException
```

Methods

The `SecurityProviderNotFoundException` class provides the following methods:

Method	Description
`equals`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.
`getMessage`	Inherited from `java.lang.Throwable`.

Method	Description
`getStackTrace`	Inherited from `java.lang.Throwable`.
`initCause`	Inherited from `java.lang.Throwable`.
`printStackTrace`	Inherited from `java.lang.Throwable`.
`setStackTrace`	Inherited from `java.lang.Throwable`.
`toString`	Inherited from `java.lang.Throwable`.

Discussion

This exception is typically thrown when the single-sign-on service is not installed.

Constructor: SecurityProviderNotFoundException

Creates a new `SecurityProviderNotFoundException` object.

Syntax

```
SecurityProviderNotFoundException()
SecurityProviderNotFoundException(
      String message
)
```

Parameters

`message` The string to return with the exception.

Class: SecurityProviderNotProvisionedException

An exception that indicates the security provider is not provisioned.

Syntax

```
class SecurityProviderNotProvisionedException
```

Methods

The `SecurityProviderNotProvisionedException` class provides the following methods:

Method	Description
`equals`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.

Method	Description
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.
`getMessage`	Inherited from `java.lang.Throwable`.
`getStackTrace`	Inherited from `java.lang.Throwable`.
`initCause`	Inherited from `java.lang.Throwable`.
`printStackTrace`	Inherited from `java.lang.Throwable`.
`setStackTrace`	Inherited from `java.lang.Throwable`.
`toString`	Inherited from `java.lang.Throwable`.

Discussion

This exception is typically thrown when the customer ID is not set.

Constructor: SecurityProviderNotProvisionedException

Creates a new `SecurityProviderNotProvisionedException` object.

Syntax

```
SecurityProviderNotProvisionedException()
SecurityProviderNotProvisionedException(
      String message
)
```

Parameters

`message` The string to return with the exception.

Class: SecurityToken

Gets and sets the information in a SAML response.

Syntax

```
class SecurityToken
```

Methods

The `SecurityToken` class provides the following methods:

Method	Description
`getAttributes`	Returns the set of attributes in the `SecurityToken` object.
`getAudience`	Returns the address to which the response has been sent.
`getAuthenticationMethod`	Returns the authentication method used for the SAML response.
`getExpiredTime`	Returns the time at which the SAML response will have expired.
`getIssuer`	Returns the issuer of the SAML response.
`getNameFormat`	Returns the format of the subject name.
`getRecipient`	Returns the address of the intended recipient of the SAML response.
`getSignatureType`	Returns the type of signature used for the SAML response.
`getSubjectConfirmationMethod`	Returns the subject confirmation method specified in the SAML response.
`getSubjectName`	Returns the subject name in the SAML response.
`getVersion`	Returns the version number of the SAML response.
`setAttributes`	Sets attributes in the `SecurityToken` object.
`setAudience`	Sets the address to which the response should be sent.
`setAuthenticationMethod`	Sets the authentication method used for the SAML response.
`setExpiredTime`	Sets the time at which the SAML response expires.
`setIssuer`	Sets the issuer name in the SAML response.
`setNameFormat`	Sets the format of the subject name.
`setRecipient`	Sets the recipient of the SAML response.
`setSignatureType`	Sets the type of signature used for the SAML response.
`setSubjectConfirmationMethod`	Sets the confirmation method used for the subject of the SAML response.
`setSubjectName`	Sets the subject name for the SAML response.
`setVersion`	Sets the version number of the SAML response.
`toString`	Concatenates all the values in the `SecurityToken` object into a single string.
`equals`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Description

The `EnterpriseAuthentication.parseSecurityToken` method parses all the information in a SAML response and returns a `SecurityToken` object. You can use the methods in this class to get and set the values in this object.

See Also

getSecurityToken

parseSecurityToken

Constructor: SecurityToken

Creates a new `SecurityToken` object.

Syntax

```
SecurityToken()
SecurityToken(
      SecurityTokenDetailsResponse rsp
)
```

Parameters

rsp The keys and values to go into the object.

Method: getAttributes

Returns the set of attributes in the `SecurityToken` object.

Class

`SecurityToken`

Syntax

```
Map<String,String> getAttributes()
```

Return value

The set of key-value pairs of all the attributes in the SAML response.

Discussion

The SAML response contains the attributes that are specified by an application-specific SAML script run by Centrify Cloud Service when the user opens the single-sign-on SaaS or mobile application. The script is created by the administrator when configuring the Generic SAML template in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

Example

The following code sample illustrates the use of the `SecurityToken.getAttributes` method:

```
protected String formatToken4Display() {
    StringBuilder sb = new StringBuilder();
    sb.append("---- Token content ----\n");
    if (token == null) {
        sb.append("Empty token object\n");
    } else {
        sb.append("subjectName:" + token.getSubjectName() + "\n");
        sb.append("audience:" + token.getAudience() + "\n");
        sb.append("issuer:" + token.getIssuer() + "\n");
        sb.append("recipient:" + token.getRecipient() + "\n");
        sb.append("version:" + token.getVersion() + "\n");
        Map<String, String> attributes = token.getAttributes();
        for (String key : attributes.keySet()) {
            sb.append(key + ":" + attributes.get(key) + "\n");
        }
    }

    sb.append("---- Token blob ----\n");
    sb.append(samlToken);
    return sb.toString();
}
```

Method: getAudience

Returns the address to which the response has been sent.

Class

`SecurityToken`

Syntax

`String getAudience()`

Return value

The URI of the audience to which the response is addressed. The audience is in the SAML assertion script configured in Centrify Cloud Manager.

Discussion

If the audience is not the current application, you should forward the SAML response to the URI specified as the audience.

The SAML assertion script is created by the administrator when configuring the Generic SAML template in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

Method: getAuthenticationMethod

Returns the authentication method used for the SAML response.

Class

`SecurityToken`

Syntax

```
String getAuthenticationMethod()
```

Return value

The authentication method as recorded in the authentication context of the SAML response.

Method: getExpiredTime

Returns the time at which the SAML response will have expired.

Class

`SecurityToken`

Syntax

```
Date getExpiredTime()
```

Return value

The GMT time at which the SAML response will have expired.

Method: getIssuer

Returns the issuer of the SAML response.

Class

`SecurityToken`

Syntax

```
String getIssuer()
```

Return value

The issuer name from the SAML response.

Discussion

The SAML response includes an `<Issuer>` element, which identifies the entity that issued the response. This method returns the name of the issuer from the `<Issuer>` element. You set the issuer in the Issuer field of the Application Settings tab when configuring the Generic SAML template for the application in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

Method: getNameFormat

Returns the format of the subject name.

Class

`SecurityToken`

Syntax

```
String getNameFormat()
```

Return value

The format used for the subject name.

Discussion

For name formats, see *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*; OASIS Standard, 15 March 2005; at http://docs.oasis-open.org/security/saml/v2.0/.

See Also

getSubjectName

Method: getRecipient

Returns the address of the intended recipient of the SAML response.

Class

`SecurityToken`

Syntax

```
String getRecipient()
```

Return value

The URI of the recipient to which the response is addressed.

Discussion

If the recipient is not the current address, you should forward the SAML response to the URI specified as the recipient. You set the recipient in the SAML assertion script in the **Advanced** tab of the Generic SAML template for the application in Centrify Cloud Manager. See "setRecipient" on page 242.

Method: getSignatureType

Returns the type of signature used for the SAML response.

Class

```
SecurityToken
```

Syntax

```
String getSignatureType()
```

Return value

The type of digital signature used. Possible values are `'Response'` or `'Assertion'`.

Discussion

SAML responses are digitally signed to ensure their integrity. The signature type specifies whether the SAML assertion itself or the response is signed. You set the signature type in the SAML assertion script in the Advanced tab of the Generic SAML template for the application in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

Method: getSubjectConfirmationMethod

Returns the subject confirmation method specified in the SAML response.

Class

```
SecurityToken
```

Syntax

```
String getSubjectConfirmationMethod()
```

Return value

The method used to confirm the subject.

Discussion

The subject confirmation method is a URI reference that identifies the method used to confirm the subject. SAML-defined confirmation methods are defined in *Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0*; OASIS Standard, 15 March 2005; at http://docs.oasis-open.org/security/saml/v2.0/.

Method: getSubjectName

Returns the subject name in the SAML response.

Class

```
SecurityToken
```

Syntax

```
String getSubjectName()
```

Return value

The name of the subject.

Discussion

The subject name specifies the principal that is the subject of the response. This is typically the Active Directory login name of the user that is being authenticated by this SAML response. You set the subject name in the SAML assertion script in the Advanced tab of the Generic SAML template for the application in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

See Also

getNameFormat

Method: getVersion

Returns the version number of the SAML response.

Class

`SecurityToken`

Syntax

```
int getVersion()
```

Return value

The major and minor version of the SAML response in the form *`major.minor`*; for example, `1.2`.

Method: setAttributes

Sets attributes in the `SecurityToken` object.

Class

`SecurityToken`

Syntax

```
void setAttributes(
      Map<String,String> attributes
)
```

Parameters

attributes The set of key-value pairs of attributes you want to set in the SAML response.

Method: setAudience

Sets the address to which the response should be sent.

Class

SecurityToken

Syntax

```
void setAudience(
      String audience
)
```

Parameters

audience The URI of the audience to which the response is to be sent.

Method: setAuthenticationMethod

Sets the authentication method used for the SAML response.

Class

SecurityToken

Syntax

```
void setAuthenticationMethod(
      String authenticationMethod
)
```

Parameters

authenticationMethod The authentication method used.

Method: setExpiredTime

Sets the time at which the SAML response expires.

Class

SecurityToken

Syntax

```
void setExpiredTime(
      Date ExpiredTime
)
```

Parameters

ExpiredTime The expiration time.

Method: setIssuer

Sets the issuer name in the SAML response.

Class

SecurityToken

Syntax

```
void setIssuer(
      String issuer
)
```

Parameters

issuer The value of the Issuer element.

Method: setNameFormat

Sets the format of the subject name.

Class

SecurityToken

Syntax

```
void setNameFormat(
      String nameFormat
)
```

Parameters

nameFormat The format of the subject name.

Method: setRecipient

Sets the recipient of the SAML response.

Class

SecurityToken

Syntax

```
void setRecipient(
      String Recipient
)
```

Parameters

Recipient The recipient.

Method: setSignatureType

Sets the type of signature used for the SAML response.

Class

SecurityToken

Syntax

```
void setSignatureType(
      String signatureType
)
```

Parameters

signatureType The signature type.

Method: setSubjectConfirmationMethod

Sets the confirmation method used for the subject of the SAML response.

Class

SecurityToken

Syntax

```
void setSubjectConfirmationMethod(
      String subjectConfirmationMethod
)
```

Parameters

subjectConfirmationMethod The confirmation method.

Method: setSubjectName

Sets the subject name for the SAML response.

Class

SecurityToken

Syntax

```
void setSubjectName(
      String subjectName
)
```

Parameters

subjectName The name of the subject of the SAML response.

Method: setVersion

Sets the version number of the SAML response.

Class

`SecurityToken`

Syntax

```
void setVersion(
      int version
)
```

Parameters

version The major and minor version numbers; for example, `1.3`.

Method: toString

Concatenates all the values in the `SecurityToken` object into a single string.

Class

`SecurityToken`

Syntax

```
void toString()
```

Discussion

This method overrides the `Object.toString` method and concatenates all of the values in the `SecurityToken` object into a single string of labeled newline-separated items in the following order:

```
version= version
issuer= issuer
signatureType= signatureType
subjectName= subjectName
nameFormat= nameFormat
recipient= recipient
subjectConfirmationMethod= subjectConfirmationMethod
expiredTime= expiredTime.toString
audience= audience
authenticationMethod= authenticationMethod
attributes= attributes.toString
```

Class: SecurityTokenInvalidException

An exception that indicates that the SAML response is not valid.

Syntax

```
class SecurityTokenInvalidException
```

Methods

The `SecurityTokenInvalidException` class provides the following methods:

Method	Description
`equals`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.
`getMessage`	Inherited from `java.lang.Throwable`.
`getStackTrace`	Inherited from `java.lang.Throwable`.
`initCause`	Inherited from `java.lang.Throwable`.
`printStackTrace`	Inherited from `java.lang.Throwable`.
`setStackTrace`	Inherited from `java.lang.Throwable`.
`toString`	Inherited from `java.lang.Throwable`.

Discussion

This exception indicates that the SAML response has either been corrupted or tampered with.

Constructor: SecurityTokenInvalidException

Creates a new `SecurityTokenInvalidException` object.

Syntax

```
SecurityTokenInvalidException()
SecurityTokenInvalidException(
```

```
      String message
)
```

Parameters

message The string to return with the exception.

Class: TargetApplicationNotSupportedException

An exception that indicates that a specified application is not supported by this installation.

Syntax

```
class TargetApplicationNotSupportedException
```

Methods

The `TargetApplicationNotSupportedException` class provides the following methods:

Method	Description
`equals`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.
`getMessage`	Inherited from `java.lang.Throwable`.
`getStackTrace`	Inherited from `java.lang.Throwable`.
`initCause`	Inherited from `java.lang.Throwable`.
`printStackTrace`	Inherited from `java.lang.Throwable`.
`setStackTrace`	Inherited from `java.lang.Throwable`.
`toString`	Inherited from `java.lang.Throwable`.

Discussion

In order to generate a SAML response, an application first has to be defined in Centrify Cloud Manager. This exception indicates that a SAML response was requested when there was no definition for the application in Centrify Cloud Manager for this user.

Constructor: TargetApplicationNotSupportedException

Creates a new `TargetApplicationNotSupportedException` object.

Syntax

```
TargetApplicationNotSupportedException()
TargetApplicationNotSupportedException(
      String message
)
```

Parameters

`message` The string to return with the exception.

Class: UserAttributes

Gets a user's Active Directory attributes such as name and address.

Syntax

```
class UserAttributes
```

Methods

The `UserAttributes` class provides the following methods:

Method	Description
`getAdditionalAttributes`	Returns attributes additional to the default set.
`getCity`	Returns the user's `City` attribute.
`getCountry`	Returns the user's `Country` attribute.
`getDescription`	Returns the user's `Description` attribute.
`getDisplayName`	Returns the user's `DisplayName` attribute.
`getEmail`	Returns the user's `Email` attribute.
`getFirstName`	Returns the user's `FirstName` attribute.
`getGroups`	Returns the groups to which the user belongs.
`getInitials`	Returns the user's `Initials` attribute.
`getLastName`	Returns the user's `LastName` attribute.
`getPostalCode`	Returns the user's `PostalCode` attribute.
`getState`	Returns the user's `State` attribute.
`getStreetAddress`	Returns the user's `StreetAddress` attribute.
`getTelephone`	Returns the user's `Telephone` attribute.
`toString`	Concatenates all the user's attributes into a single string.

Method	Description
`equals`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

The `EnterpriseAuthentication.getUserInformation` method returns a `userAttributes` object containing the values of Active Directory attributes for the authenticated user. You can use the "get" methods in this class to get the attribute values.

See Also

getUserInformation

Constructor: UserAttributes

Creates a new `UserAttributes` object.

Syntax

```
UserAttributes(
      UserLookupResult lookupResult
)
```

Parameters

`lookupResult` The user object returned from the authentication provider.

Method: getAdditionalAttributes

Returns attributes additional to the default set.

Class

`UserAttributes`

Syntax

```
Map<String,String[]> getAdditionalAttributes()
```

Return value

The user's additional attributes as a map of attribute names and values.

Discussion

Additional attributes, if any, are added to the `UserAttributes` object by the `EnterpriseAuthentication.userLookup` method.

See Also

userLookup

Method: getCity

Returns the user's `City` attribute.

Class

`UserAttributes`

Syntax

```
String getCity()
```

Return value

The user's `City` attribute.

Method: getCountry

Returns the user's `Country` attribute.

Class

`UserAttributes`

Syntax

```
String getCountry()
```

Return value

The user's `Country` attribute.

Method: getDescription

Returns the user's `Description` attribute.

Class

`UserAttributes`

Syntax

```
String getDescription()
```

Return value

The user's `Description` attribute.

Method: getDisplayName

Returns the user's `DisplayName` attribute.

Class

`UserAttributes`

Syntax

```
String getDisplayName()
```

Return value

The user's `DisplayName` attribute.

Method: getEmail

Returns the user's `Email` attribute.

Class

`UserAttributes`

Syntax

```
String getEmail()
```

Return value

The user's `Email` attribute.

Method: getFirstName

Returns the user's `FirstName` attribute.

Class

`UserAttributes`

Syntax

```
String getFirstName()
```

Return value

The user's `FirstName` attribute.

Method: getGroups

Returns the groups to which the user belongs.

Class

`UserAttributes`

Syntax

```
String[] getGroups()
```

Return value

An array of groups to which the user belongs.

Method: getInitials

Returns the user's `Initials` attribute.

Class

`UserAttributes`

Syntax

```
String getInitials()
```

Return value

The user's `Initials` attribute.

Method: getLastName

Returns the user's `LastName` attribute.

Class

`UserAttributes`

Syntax

```
String getLastName()
```

Return value

The user's `LastName` attribute.

Method: getPostalCode

Returns the user's `PostalCode` attribute.

Class

`UserAttributes`

Syntax

```
String getPostalCode()
```

Return value

The user's `PostalCode` attribute.

Method: getState

Returns the user's `State` attribute.

Class

`UserAttributes`

Syntax

```
String getState()
```

Return value

The user's `State` attribute.

Method: getStreetAddress

Returns the user's `StreetAddress` attribute.

Class

`UserAttributes`

Syntax

```
String getStreetAddress()
```

Return value

The user's `StreetAddress` attribute.

Method: getTelephone

Returns the user's `Telephone` attribute.

Class

`UserAttributes`

Syntax

```
String getTelephone()
```

Return value

The user's `Telephone` attribute.

Method: toString

Concatenates all the user's attributes into a single string.

Class

`UserAttributes`

Syntax

```
void toString()
```

Discussion

This method overrides the `Object.toString` method and concatenates all of the user's attributes into a single string of labeled newline-separated items in the following order:

`displayName=` *`displayName`*
`description=` *`description`*
`email=` *`email`*
`firstName=` *`firstName`*
`lastName=` *`lastName`*
`initials=` *`initials`*
`telephoneNumber=` *`telephoneNumber`*
`streetAddress=` *`streetAddress`*
`city=` *`city`*
`state=` *`state`*
`country=` *`state`*
`postalCode=` *`postalCode`*
`groups=` *`groups`* (as a list of groups with a comma and space character separating each item)

See Also

getAdditionalAttributes

Class: UserInformation

Gets a user's authentication information.

Syntax

```
class UserInformation
```

Methods

The `UserInformation` class provides the following methods:

Method	Description
`getAuthType`	Returns the type of authentication being used.
`getLastAuthMillis`	Returns the time of the most recent authentication.
`getUserName`	Returns the user name used for authentication.
`setAuthType`	Sets the type of authentication in use.
`setLastAuthMillis`	Sets the time of the most recent authentication.
`setUserName`	Sets the user name used for authentication.
`equals`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

The `EnterpriseAuthentication.getUserInformation` method returns a `userInformation` object containing the values of authentication attributes for the authenticated user from Centrify Cloud Service. You can use the "get" methods in this class to get the attribute values. You can use the "set" methods to set the values of attributes in the local `userInformation` object.

See Also

userLookup

Constructor: UserInformation

Creates a new `UserInformation` object.

Syntax

```
UserInformation()
```

Method: getAuthType

Returns the type of authentication being used.

Class

`UserInformation`

Syntax

```
String getAuthType()
```

Return value

The authentication type.

Method: getLastAuthMillis

Returns the time of the most recent authentication.

Class

`UserInformation`

Syntax

```
long getLastAuthMillis()
```

Return value

The last time at which the user was authenticated, in milliseconds since 00:00:00 GMT, 1 January, 1970.

Example

The following code sample illustrates the use of the `UserInformation.getLastAuthMillis` method:

```
protected void onPostExecute(UserInformation result) {
    super.onPostExecute(result);
    Log.i(TAG, "User info loaded, result:" + result);

    if (result == null) {
        mBodyText.setText("Authentication failed:" + error);
        return;
    }

    SimpleDateFormat dateFormat = new SimpleDateFormat(
        "EEE, d MMM yyyy HH:mm:ss");
    final String msg = String.format("Welcome: %s\nLast auth time: %s",
        result.getUserName(),
        dateFormat.format(new Date(result.getLastAuthMillis())));
    mBodyText.setText(msg);
}
```

Method: getUserName

Returns the user name used for authentication.

Class

`UserInformation`

Syntax

```
String getUserName()
```

Return value

The user name of the authenticated user.

Example

The following code sample illustrates the use of the `UserInformation.getUserName` method:

```
protected void onPostExecute(UserInformation result) {
    super.onPostExecute(result);
    Log.i(TAG, "User info loaded, result:" + result);

    if (result == null) {
        mBodyText.setText("Authentication failed:" + error);
        return;
    }

    SimpleDateFormat dateFormat = new SimpleDateFormat(
        "EEE, d MMM yyyy HH:mm:ss");
    final String msg = String.format("Welcome: %s\nLast auth time: %s",
        result.getUserName(),
        dateFormat.format(new Date(result.getLastAuthMillis())));
    mBodyText.setText(msg);
}
```

Method: setAuthType

Sets the type of authentication in use.

Class

`UserInformation`

Syntax

```
void setAuthType(
      String authType
)
```

Parameters

`authType` The authentication type.

Method: setLastAuthMillis

Sets the time of the most recent authentication.

Class

`UserInformation`

Syntax

```
String setLastAuthMillis(
      long lastAuthMillis
)
```

Parameters

lastAuthMillis The time of the most recent authentication, in milliseconds since 00:00:00 GMT, 1 January, 1970.

Method: setUserName

Sets the user name used for authentication.

Class

```
UserInformation
```

Syntax

```
String setUserName(
      String userName
)
```

Parameters

userName The user name of the authenticated user.

Android MAS Sample Code

As illustrated in Figure 1. Centrify cloud components, there are four main components involved in Centrify Cloud Service:

- A corporate or organization's network running the Centrify Cloud Proxy Server and Active Directory.
- Centrify cloud authentication service.
- One or more providers of Internet (that is, cloud) based software (software as a service, or SaaS) including web applications that may be on-premises or hosted remotely.
- Mobile or desktop devices that need to access the web applications, where the users of the devices belong to groups and roles set up in Active Directory.

Centrify Cloud Service provides authentication and single sign-on (SSO) services for the mobile devices and for the SaaS application.

This chapter walks through the Java code for a sample mobile application that uses the Android MAS API.

MainActivity

The initial file of the Android SSO Demo1 sample application performs the following actions:

1. Imports Android libraries. Step 1
2. Initializes the interface with saved state information. Step 2
3. Sets up default settings. Step 3
4. Initializes the Options menu by instantiating a menu XML file into a menu object. The menu XML file is in GenericSSODemo1 > res > menu > settings.xml. Step 4
5. Initiates an action when the user selects an item in the Options menu. Step 5
6. Branches to an action depending on which view the user clicks. Step 6
7. Executes the code in the file `GetUserInfoActivity.java` (see GetUserInfoActivity). Step 7
8. Executes the code in the file `SalesRecordsActivity.java` (see SalesRecordsActivity). Step 8
9. Executes the code in the file `UserLookupActivity.java` (see UserLookupActivity) Step 9

```
package com.centrify.auth.demo1;
/*
Step 1
*/
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

public class MainActivity extends Activity {
/*
Step 2
Called when the activity is first created.
*/
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
/*
Step 3
*/
        Preferences.setupDefaults(this);
    }
/*
Step 4
*/
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.settings, menu);
        return true;
    }
/*
Step 5
*/
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (item.getItemId() == R.id.settings_menu) {
            Intent intent = new Intent(this, SettingsActivity.class);
            startActivity(intent);
            return true;
        } else {
            return super.onOptionsItemSelected(item);
        }
    }
/*
Step 6
*/
    public void onClick(View view) {
        switch (view.getId()) {
/*
Step 7
*/
        case R.id.btnGetUserInfo:
            GetUserInfoActivity.startActivity(this);
            break;
/*
Step 8
*/
        case R.id.btnGetGetSecurityToken:
            SalesRecordsActivity.startActivity(this);
            break;
/*
```

```
Step 9
*/
        case R.id.btnUserLookup:
            UserLookupActivity.startActivity(this);
            break;

        default:
            break;
        }
    }

}
```

GetUserInfoActivity

The `GetUserInfoActivity.java` file of the Android SSO Demo1 sample application illustrates the use of the methods of the Android MAS API to get information about the authenticated user. The code, shown below, performs the following actions:

1. Imports Java libraries. Step 1
2. Imports Centrify libraries. Step 2
3. Imports Android libraries. Step 3
4. Defines a new class, `GetUserInfoActivity`, by extending the Android `Activity` class. Step 4
5. Initializes the interface with saved state information. Step 5
6. Activates the lookup activity when it is in front. Step 6
7. Gets the user information in the background as an asynchronous task. Step 7
8. Creates a new `EnterpriseAuthentication` object (see "EnterpriseAuthentication" on page 56), which can be used to return user information and the SAML response, and calls `EnterpriseAuthentication.getUserInformation` to get a `UserInformation` object (see "UserInformation" on page 87) that contains the currently authenticated user's logon name, the authentication type, and the time in milliseconds of the last authentication. If the user is not already authenticated, Centrify Cloud Services displays an authentication dialog requesting the user's Active Directory credentials. This is a blocking call, so it does not return until the user has responded. Step 8
9. Logs the result of the call: either the information was returned or authentication failed. Step 9
10. Specifies the format for the date. Step 10
11. Calls `UserInformation.getUserName` and `UserInformation.getLastAuthMillis` to get the user name and authentication time and displays the results. Step 11

```
package com.centrify.sso.demo1;
/*
Step 1
```

```
*/
import java.text.SimpleDateFormat;
import java.util.Date;
/*
Step 2
*/
import com.centrify.auth.EnterpriseAuthentication;
import com.centrify.auth.UserInformation;
/*
Step 3
*/
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
/*
Step 4
*/
public class GetUserInfoActivity extends Activity {
    protected static final String TAG = "GetUserInfoActivity";

    private AsyncTask<Void, Void, UserInformation> loadTask;
    private TextView bodyText;

    public static void startActivity(Context ctx) {
        ctx.startActivity(new Intent(ctx, GetUserInfoActivity.class));
    }

/*
Step 5
    Called when the activity is first created.
*/
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.get_user_info);
        bodyText = (TextView)this.findViewById(R.id.bodyText);
    }
/*
Step 6
*/
    @Override
    protected void onResume() {
        super.onResume();
        if (loadTask == null) {
            getUserInfo();
        }
    }
/*
Step 7
*/
    private void getUserInfo() {
        Log.i(TAG, "Enter getUserInfo");
        loadTask = new AsyncTask<Void, Void, UserInformation>() {
            private String error;

            @Override
            protected UserInformation doInBackground(Void... params) {
/*
Step 8
*/
                try {
```

```
                    return EnterpriseAuthentication.getUserInformation
                        (GetUserInfoActivity.this);
                } catch (Exception e) {
                    Log.w(TAG, "Authentication failed:" + e);
                    error = e.toString();
                }
                // Auth failed
                return null;
            }
/*
Step 9
*/
            @Override
            protected void onPostExecute(UserInformation result) {
                super.onPostExecute(result);
                Log.i(TAG, "User info loaded, result:" + result);

                if (result == null) {
                    bodyText.setText("Authentication failed:" + error);
                    return;
                }
*/
Step 10
/*
                SimpleDateFormat dateFormat =
                    new SimpleDateFormat("EEE, d MMM yyyy HH:mm:ss");
/*
Step 11
*/
                final String msg = String.format("Welcome: %s\nLast auth time: %s",
                    result.getUserName(),  dateFormat.format(new
                    Date(result.getLastAuthMillis())));
                bodyText.setText(msg);
            }
        };
        Log.i(TAG, "Start to load user info");
        loadTask.execute();
    }
}
```

SalesRecordsActivity

The `SalesRecordsActivity.java` file of the Android SSO Demo1 sample application illustrates the use of the methods of the Android MAS API to get a SAML response from Centrify cloud and pass it to the web service to authenticate the user and obtain data. The code, shown below, performs the following actions:

1. Imports Java libraries. Step 1
2. Imports Apache libraries. Step 2
3. Imports Json libraries. Step 3
4. Imports Android libraries. Step 4
5. Imports Centrify libraries. Step 5

6 Defines a new class, `SalesRecordsActivity`, by extending the Android `Activity` class. Step 6

7 Initializes the interface with saved state information. Step 7

8 If the token has not been downloaded already, executes the code to download the token. Step 8

9 Gets the token in the background as an asynchronous task. Step 9

10 Creates a new `EnterpriseAuthentication` object (see "EnterpriseAuthentication" on page 56), which can be used to return the SAML response and calls `EnterpriseAuthentication.getSecurityToken` to get the SAML response (see "SecurityToken" on page 67). The first parameter is the name of the application as defined in Centrify Cloud Manager. The second parameter specifies that the token does not need to be downloaded if it is already cached. The third parameter is the Android context. Step 10

If the user is not already authenticated, Centrify Cloud Services displays an authentication dialog requesting the user's Active Directory credentials. This is a blocking call, so it does not return until the user has responded.

11 Calls the `ssoRequest` method (see Step 14), which sends the SAML authentication response to the web application and requests the sales data. Step 11

12 Updates the progress indicator as the background task executes. Step 12

13 Calls the `displayRecords` method (see Step 18) to display the results when the task completes. Step 13

14 Sends the SAML authentication response to the web application, requests the sales data, and receives the results. Step 14

15 Gets the URL of the web service and begins to format a message. Step 15

16 Adds the SAML response to the message header. The code shows both use of a REST interface and a POST message to send the SAML response. Step 16

17 Receives the response and concatenates it into a string. Step 17

18 Formats and displays the results. Step 18

```
package com.centrify.sso.demo1;
/*
Step 1
*/
import java.util.ArrayList;
import java.util.List;
/*
Step 2
*/
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
```

```
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
/*
Step 3
*/
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
/*
Step 4
*/
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.webkit.WebView;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;
import android.widget.Toast;
/*
Step 5
*/
import com.centrify.auth.EnterpriseAuthentication;
import com.centrify.auth.TargetApplicationNotSupportedException;
import com.centrify.auth.demo1.http.HttpClientHelper;
/*
Step 6
*/
public class SalesRecordsActivity extends Activity {
    protected static final String TAG = "SalesRecordsActivity";

    private AsyncTask<Void, String, String> loadTask;
    private TextView bodyText;
    private String samlToken = "";
    private String ssoResult = "";

    public static void startActivity(Context ctx) {
        ctx.startActivity(new Intent(ctx, SalesRecordsActivity.class));
    }
/*
Step 7
Called when the activity is first created.
*/
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.sales_records);
        bodyText = (TextView)this.findViewById(R.id.bodyText);
    }
/*
Step 8
*/
    @Override
    protected void onStart() {
        super.onStart();
        if (loadTask == null) {
            getToken();
        }
```

```
    }
/*
Step 9
*/
    private void getToken() {
        Log.i(TAG, "Enter getToken");
        loadTask = new AsyncTask<Void, String, String>() {

            @Override
            protected String doInBackground(Void... params) {
/*
Step 10
*/
                try {
                    String target = Preferences.getTarget(SalesRecordsActivity.this);
                    samlToken = EnterpriseAuthentication.getSecurityToken
                        (target, false, SalesRecordsActivity.this);
                    if (samlToken == null || samlToken.length() == 0) {
                        Toast.makeText(SalesRecordsActivity.this, "No saml token found",
                            Toast.LENGTH_SHORT).show();
                        return null;
                    }
                    publishProgress("Auth token obtained, start to fetch sales
                        records...");
/*
Step 11
*/
                    ssoRequest();

                } catch (TargetApplicationNotSupportedException e) {
                    publishProgress("Sales record fetched.");
                    Log.w(TAG, "Get token failed:" + e);
                    publishProgress("Failed to obtain token:" + e);
                } catch (Exception e) {
                    Log.w(TAG, "Get token failed:" + e);
                    publishProgress("Failed to obtain token:" + e);
                }
                // Auth failed
                return null;
            }
/*
Step 12
*/
            @Override
            protected void onProgressUpdate(String... values) {
                bodyText.setText(values[0]);
            }
/*
Step 13
*/
            @Override
            protected void onPostExecute(String result) {
                super.onPostExecute(result);
                if (Preferences.getMethod(SalesRecordsActivity.this).
                    equals(Preferences.METHOD_REST)) {
                    displayRestRecords();
                } else if (Preferences.getMethod(SalesRecordsActivity.this).
                    equals(Preferences.METHOD_POST)) {
                    displayPostRecords();
                }
            }
        };
        Log.i(TAG, "Start to load sales records");
        loadTask.execute();
    }
```

```
/*
Step 14
*/
    private void ssoRequest() {
        DefaultHttpClient httpClient = HttpClientHelper.createHttpClient(true);
/*
Step 15
*/
        try {
            String url = Preferences.getRemoteServer(this);
            HttpUriRequest request = null;
/*
Step 16
*/
            if (Preferences.getMethod(this).equals(Preferences.METHOD_REST)) {
                HttpGet httpGet = new HttpGet(url);
                httpGet.addHeader("SAMLResponse", samlToken);
                request = httpGet;
            } else if (Preferences.getMethod(this).equals(Preferences.METHOD_POST)) {
                HttpPost httpPost = new HttpPost(url);
                List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(1);
                nameValuePairs.add(new BasicNameValuePair("SAMLResponse", samlToken));
                httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
                request = httpPost;
            }

            HttpResponse httpResponse = httpClient.execute(request);/*
/*
Step 17
*/
            if (httpResponse.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
                ssoResult = EntityUtils.toString(httpResponse.getEntity());
            } else {
                Log.w(TAG, "Http request failed, " +
                    httpResponse.getStatusLine().toString());
                ssoResult = httpResponse.getStatusLine().toString();
                httpResponse.getEntity().consumeContent();
            }
            Log.d(TAG, "getcustomers response:\n" + ssoResult);
        } catch (Exception ex) {
            Log.w(TAG, "Failed to send sso request" , ex);
        }
    }
/*
Step 18
*/

    private void displayPostRecords() {
        WebView view = (WebView)findViewById(R.id.html_view);
        view.setVisibility(View.VISIBLE);
        if (ssoResult != null) {
            view.loadData(ssoResult, "text/html", "utf-8");
        }
    }

    private void displayRestRecords() {
/*
The data returned by this sample application is:
S[{"Name":"AcmeWidgets","SalesMan":null,"Sales":20},{"Name":"BigMines","SalesMan":null,
    "Sales":40},{"Name":"FastPlanes","SalesMan":null,"Sales":100}]
*/
        Log.d(TAG, "Display records");
        TableLayout table = (TableLayout)findViewById(R.id.salesRecordTable);
        table.setVisibility(View.VISIBLE);
        try {
```

```
                JSONObject resultObj = new JSONObject(ssoResult);

                boolean success = resultObj.getBoolean("success");

                // Message
                String message = resultObj.getString("message");
                TextView messageView = new TextView(this);
                messageView.setText(message);
                TableRow row = new TableRow(this);
                row.addView(messageView);
                table.addView(row);

                if (success) {
                    // Title
                    TableRow titleRow = new TableRow(this);
                    TextView columnView1 = new TextView(this);
                    columnView1.setText("Customer");
                    titleRow.addView(columnView1);
                    TextView columnView2 = new TextView(this);
                    columnView2.setText("Sales");
                    titleRow.addView(columnView2);
                    table.addView(titleRow);
                    titleRow.setBackgroundColor(Color.YELLOW);

                    JSONArray recordsObj = resultObj.getJSONArray("payload");
                    for (int i = 0; i < recordsObj.length(); i++) {
                        JSONObject record = new JSONObject(recordsObj.getString(i));
                        String name = record.getString("Name");
                        String number = record.getString("Sales");
                        row = new TableRow(this);
                        row.setBackgroundColor(i % 2 == 0 ? Color.GRAY : Color.LTGRAY);
                        TextView nameView = new TextView(this);
                        nameView.setText(name);
//                      nameView.setLayoutParams(new LayoutParams(100, 100));
                        row.addView(nameView);

                        TextView numberView = new TextView(this);
                        numberView.setText(number);
//                      numberView.setLayoutParams(new LayoutParams(100, 100));
                        row.addView(numberView);

                        table.addView(row);
                    }
                }
            } catch (JSONException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();

                TableRow row = new TableRow(this);
                TextView messageView = new TextView(this);
                messageView.setText(ssoResult);
                row.addView(messageView);
                table.addView(row);
            }
        }
    }
```

UserLookupActivity

The `UserLookupActivity.java` file of the Android SSO Demo1 sample application illustrates the use of the methods of the Android MAS API to get Active Directory attributes for the authenticated user. The code, shown below, performs the following actions:

1. Imports Android libraries. Step 1
2. Imports Centrify libraries. Step 2
3. Defines a new class, `UserLookupActivity`, by extending the Android `Activity` class. Step 3
4. Initializes the interface with saved state information. Step 4
5. Activates the lookup activity when it is in front. Step 5
6. Gets the user information in the background as an asynchronous task. Step 6
7. Creates a new `EnterpriseAuthentication` object (see "EnterpriseAuthentication" on page 56), which can be used to return user information, and calls `EnterpriseAuthentication.userLookup` to get a `UserAttributes` object (see "UserAttributes" on page 81) that contains the currently authenticated user's Active Directory `proxyaddresses` attribute. If the user is not already authenticated, Centrify Cloud Services displays an authentication dialog requesting the user's Active Directory credentials. This is a blocking call, so it does not return until the user has responded.. Step 7

   Note The SAML response contains the attributes that are specified by an application-specific SAML script run by Centrify Cloud Service. The script is created by the administrator when configuring the Generic SAML template in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.
8. Logs the result of the call: either the information was returned or authentication failed. Step 8
9. Displays the attribute as a string. Step 9

```
package com.centrify.sso.demo1;
/*
Step 1
*/
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
/*
Step 2
*/
```

```
import com.centrify.auth.EnterpriseAuthentication;
import com.centrify.auth.UserAttributes;/*
Step 3
*/
public class UserLookupActivity extends Activity {
    protected static final String TAG = "UserLookupActivity";

    private AsyncTask<Void, Void, UserAttributes> loadTask;
    private TextView bodyText;

    public static void startActivity(Context ctx) {
        ctx.startActivity(new Intent(ctx, UserLookupActivity.class));
    }


/*
Step 4
/** Called when the activity is first created.
*/
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user_lookup);
        bodyText = (TextView)this.findViewById(R.id.bodyText);
    }
/*
Step 5
*/
    @Override
    protected void onResume() {
        super.onResume();
        if (loadTask == null) {
            getUserAttributes();
        }
    }
/*
Step 6
*/
    private void getUserAttributes() {
        Log.i(TAG, "Enter userLookup");
        loadTask = new AsyncTask<Void, Void, UserAttributes>() {
            private String error;

            @Override
            protected UserAttributes doInBackground(Void... params) {
/*
Step 7
*/
                try {
                    String[] attributes = new String[] { "proxyaddresses" };
                    return EnterpriseAuthentication.userLookup(UserLookupActivity.this,
                            attributes);
                } catch (Exception e) {
                    Log.w(TAG, "Authentication failed:" + e);
                    error = e.toString();
                }
                // Auth failed
                return null;
            }
/*
Step 8
*/
            @Override
            protected void onPostExecute(UserAttributes result) {
                super.onPostExecute(result);
```

```
                Log.i(TAG, "User info loaded, result:" + result);

                if (result == null) {
                    bodyText.setText("Authentication failed:" + error);
                    return;
                }
/*
Step 9
*/
                bodyText.setText(result.toString());
            }
        };
        Log.i(TAG, "Start to load user attributes");
        loadTask.execute();
    }
}
```

Chapter 8

Mobile Authentication Service iOS Reference

This chapter describes the classes, methods, and properties in the Centrify Mobile Authentication Service iOS API. This chapter assumes you are familiar with the syntax of the Objective-C programming language, as described in the document Programming with Objective-C.

The classes for working with Centrify Mobile Authentication Service are defined in the `CentrifySDK.framework` framework and consist of:

Class	Description
`EnterpriseAuthentication`	Returns authentication information and attributes for a specified user and a SAML response for a specified application.
`CentrifySDKError`	Returns an error or exception for a method in this API.
`CentrifySecurityToken`	Provides access to the information in a parsed SAML response.
`CentrifySDKResult`	Encapsulates the result of a call to an API method, including the data and any error returned by the call.
`UserAttributes`	Holds a user's Active Directory attributes such as name and address.
`UserInformation`	Holds a user's authentication information.

Class: EnterpriseAuthentication

Returns authentication information and attributes for a specified user and a SAML response for a specified application.

Syntax

```
@interface EnterpriseAuthentication : NSObject
```

Methods

The `EnterpriseAuthentication` class provides the following methods:

Method	Description
`getSecurityTokenForTarget: alwaysUseFreshToken: completionHandler:`	Returns the SAML response of the specified application.
`getUserInformation:`	Returns authentication information about the currently authenticated user.

Method	Description
`instance`	Returns the current `EnterpriseAuthentication` object.
`logout:`	Resets the user's authentication session.
`parseSecurityToken:`	Parses the SAML response to create a `CentrifySecurityToken` object.
`userLookupForAdditionalAttributes:completionHandler:`	Returns the attributes of the currently authenticated user.
`class`	Inherited from `NSObject`.
`conformsToProtocol:`	Inherited from `NSObject`.
`description`	Inherited from `NSObject`.
`debugDescription`	Inherited from `NSObject`.
`hash`	Inherited from `NSObject`.
`isEqual`	Inherited from `NSObject`.
`isKindOfClass`	Inherited from `NSObject`.
`isMemberOfClass`	Inherited from `NSObject`.
`isProxy:`	Inherited from `NSObject`.
`performSelector:`	Inherited from `NSObject`.
`performSelector:withObject:`	Inherited from `NSObject`.
`performSelector:withObject:withObject:`	Inherited from `NSObject`.
`respondsToSelector:`	Inherited from `NSObject`.
`self:`	Inherited from `NSObject`.
`superclass:`	Inherited from `NSObject`.

Discussion

This class authenticates a user and returns information about the user from Active Directory that you can use for authorization purposes. It also returns a SAML response for a specified application that you can use to authenticate the user or can forward to a SaaS application to prove that the user is authenticated. Once the user is authenticated, the application on this device is registered with Centrify Cloud Service so that further calls to the methods of this class do not require reauthentication.

This is the main class of the API. All other classes in this API require that the methods of this class be called first to populate the properties of the other classes.

The methods of this class that require authentication are nonblocking and accept a pointer to a handler method that the method calls when the action completes.

See Also

CentrifySDKResult

UserInformation

UserAttributes

Method: getSecurityTokenForTarget: alwaysUseFreshToken: completionHandler:

Returns the SAML response of the specified application.

Class

`EnterpriseAuthentication`

Syntax

```
+(void) getSecurityTokenForTarget:(NSString *) target
      alwaysUseFreshToken:(Bool) alwaysUseFreshToken
      completionHandler:(CentrifyAPIHandler) handler
)
```

Parameters

`target` The name, application ID, or application key of the application for which you want the SAML response. The application name is assigned by the administrator in the generic SAML template and is not guaranteed to be unique. The application ID is assigned by the administrator in the **Application ID** field of the **Application Settings** tab of the Generic SAML template and must be unique within an organization. The application key is assigned by Centrify Cloud Manager and is guaranteed to be universally unique.
You can find the Application key in the Cloud Manager window of the generic SAML template in the **Identity Provider Sign-in URL** field. For example:

```
https://devdog.centrify.com/run?appkey=2ff55fa1-cce8-49ec-ad24-
199333fa33a2&customerid=HT150
```

`alwaysUseFreshToken` Set `TRUE` to require the method to download the token from the server even if a token is already cached. Set `FALSE` to use the token that's already been downloaded, if one is available.

`handler` The callback method that this method calls when it completes. The type definition for the callback method is:

```
typedef void (^ CentrifyApiHandler)(CentrifySDKResult *result);
```

Return value

The authentication token of the specified application encoded in Base64-encoded XML format, returned in the `CentrifyAPIHandler` object, which is the argument of the handler callback method. The authentication token is valid only if the error is `nil`.

Errors

The `getSecurityTokenForTarget:alwaysUseFreshToken:completionHandler:` method may return an error of type `CentrifySDKError` in the `CentrifySDKResult` object.

Discussion

If the user is not already authenticated, Centrify Cloud Service puts up an authentication dialog requesting the user to enter Active Directory credentials and the API does not call the handler until the user enters the credential or cancels.

If your mobile application has a back-end web service that uses SAML authentication, then the target is the name, ID, or key of the SaaS application. If you are setting up a standalone mobile application, then the target is the mobile application.

The method searches first for a match to the application ID, then for a match to the application name, then for a match to the application key. If the method finds duplicate application IDs or application names, the method fails.

Because the token used by Centrify Cloud Service for mobile applications is a SAML response, you must use the Generic SAML template in Centrify Cloud Manager to configure the SAML response. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

When you receive the SAML response, you can use it to authenticate the user, or you can pass it to the SAML interface of the web service, which uses the SAML response to log in the user. In the latter case, the user gets access to the web service.

Example

The following code sample illustrates the use of the `getSecurityTokenForTarget:alwaysUseFreshToken:completionHandler:` method:

```
- (IBAction)getAccessToken:(id)sender
{
    self.accessToken = nil;
    [EnterpriseAuthentication getSecurityTokenForTarget:@"sdkdemo"
        alwaysUseFreshToken:NO completionHandler:^(CentrifySDKResult *result) {
        [self getSecurityTokenHandler:result];
    }];
}
...
- (void) getSecurityTokenHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
```

```
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
    else
    {
        // auth Token received
        NSString *token = (NSString*)result.resultData;
        NSLog(@"Auth Token Received : %@", token);
        self.accessToken = token;

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"Access Token"
            message:token delegate:self cancelButtonTitle:@"OK" otherButtonTitles:@"Parse
            Token", @"Open Url", nil];
        alert.tag = AlertViewTagSecurityToken;
        [alert show];
    }
}
```

See Also

parseSecurityToken:

CentrifySecurityToken

Method: getUserInformation:

Returns authentication information about the currently authenticated user.

Class

`EnterpriseAuthentication`

Syntax

```
+ (void)getUserInformation:(CentrifyApiHandler)handler
```

Parameters

`handler` The callback method that this method calls when it completes. The type definition for the callback method is:

```
typedef void (^ CentrifyApiHandler)(CentrifySDKResult *result);
```

Return value

The currently authenticated user's logon name, the authentication type, and the time in milliseconds of the last authentication. This data is returned as a `UserInformation` object through the argument of the handler callback method. The data is valid only if the error is `nil`.

Errors

The `getUserInformation:` method may return an error of type `CentrifySDKError` in the `CentrifySDKResult` object.

Discussion

If the user is not already authenticated, Centrify Cloud Service puts up an authentication dialog requesting the user to enter Active Directory credentials and the API does not call the handler until the user enters the credential or cancels.

Example

The following code sample illustrates the use of the `getUserInformation:` method:

```
- (IBAction)getUserInformation:(id)sender
{
    [EnterpriseAuthentication getUserInformation:^(CentrifySDKResult *result) {
        [self getUserInformationHandler:result];
    }];
}
...
- (void) getUserInformationHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
    else
    {
        // User Information received
        UserInformation *userInfo = (UserInformation*)result.resultData;
        NSLog(@"User Information Received : %@", userInfo);

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"UserName"
            message:[userInfo description] delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
        [alert show];
    }
}
```

See Also

userLookupForAdditionalAttributes: completionHandler:

UserInformation

Method: instance

Returns the current `EnterpriseAuthentication` object.

Syntax

```
+ (EnterpriseAuthentication *) instance
```

Return value

The the current `EnterpriseAuthentication` object.

Method: logout:

Resets the user's authentication session.

Class

`EnterpriseAuthentication`

Syntax

```
+ (void)logout:(CentrifyAPIHandler)handler
```

Parameters

`handler` The callback method that this method calls when it completes. The type definition for the callback method is:

```
typedef void (^ CentrifyApiHandler)(CentrifySDKResult *result);
```

Errors

The `logout:` method may return an error of type `CentrifySDKError` in the `CentrifySDKResult` object. The logout is successful only if the error is `nil`.

Discussion

If the user has been authenticated, this method cancels the authentication so that the user must resubmit credentials the next time you call any method that requires authentication.

Example

The following code sample illustrates the use of the `logout:` method:

```
- (IBAction)logoutUser:(id)sender{
    [EnterpriseAuthentication logout:^(CentrifySDKResult *result) {
        [self logoutHandler:result];
    }];
}
...
- (void) logoutHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
    else
    {
        // Logout is complete
        NSLog(@"Logout successful");

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"Success"
            message:@"Logout successful" delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
        [alert show];
    }
}
```

Method: parseSecurityToken:

Parses the SAML response to create a `CentrifySecurityToken` object.

Class

`EnterpriseAuthentication`

Syntax

`+ (CentrifySDKResult *)parseSecurityToken:(NSString *)`*`token`*

Parameters

`token` The SAML response encoded in Base64-encoded XML format.

Return value

A pointer to a `CentrifySDKResult` object, which contains the `CentrifySecurityToken` object corresponding to the specified SAML response.

Errors

The `parseSecurityToken:` method may return an error of type `CentrifySDKError` in the `CentrifySDKResult` object. The SAML response is valid only if the error is `nil`.

Discussion

This method validates the digital signature of the SAML response and parses the token into a Java object. When you receive a SAML response, you can use it to authenticate the user, or you can pass it to the SAML interface of the SaaS application, which uses the response as a token to log in the user. In the former case, you can use the methods of the `SecurityToken` class to examine the contents of the response. In the latter case, pass the unparsed string version of the SAML response to the SaaS application.

Example

The following code sample illustrates the use of the `parseSecurityToken:` method:

```
- (void)alertView:(UIAlertView *)alertView
{
    CentrifySDKResult *result = [EnterpriseAuthentication
            parseSecurityToken:self.accessToken];
    if (result.error)
        {
            NSLog(@"Error occurred : %@", [result.error localizedDescription]);
            UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
                message:[result.error localizedDescription] delegate:nil
                cancelButtonTitle:@"OK" otherButtonTitles:nil];
            [alert show];
    }
    else
    {
        CentrifySecurityToken *parsedToken =
            (CentrifySecurityToken*)result.resultData;
        NSLog(@"Parsed Token : %@", [parsedToken description]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"Parsed Token"
```

```
            message:[parsedToken description] delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
        [alert show];
    }
}
```

See Also

getSecurityTokenForTarget: alwaysUseFreshToken: completionHandler:

CentrifySecurityToken

Method: userLookupForAdditionalAttributes: completionHandler:

Returns the attributes of the currently authenticated user.

Class

`EnterpriseAuthentication`

Syntax

```
+ (void)userLookupForAdditionalAttributes:(NSArray *)additionalAttributes
completionHandler:(CentrifyApiHandler)handler
```

Parameters

`additionalAttributes` The names of Active Directory attributes in addition to the default Active Directory user attributes, if any are required. Pass `nil` if no additional attributes are required.

`handler` The callback method that this method calls when it completes. The type definition for the callback method is:

```
typedef void (^ CentrifyApiHandler)(CentrifySDKResult *result);
```

Return value

The user's attributes, such as their name, address, and groups to which they are assigned. This data is returned as a `UserAttributes` object through the argument of the handler callback method. The data is valid only if the error is `nil`.

Errors

The `userLookupForAdditionalAttributes:completionHandler:` method may return an error of type `CentrifySDKError` in the `CentrifySDKResult` object.

Discussion

If the user is not already authenticated, Centrify Cloud Service puts up an authentication dialog requesting the user to enter Active Directory credentials and the API does not call the handler until the user enters the credential or cancels.

Example

The following code sample illustrates the use of the `userLookupForAdditionalAttributes:completionHandler:` method:

```
- (IBAction)lookupUserAttribute:(id)sender
{
    NSArray *attributes = [NSArray arrayWithObjects:@"postOfficeBox",
            @"wWWHomePage", nil];
    [EnterpriseAuthentication userLookupForAdditionalAttributes:attributes
            completionHandler:^(CentrifySDKResult *result) {
        [self lookupUserAttributeHandler:result];
    }];
}
...
- (void) lookupUserAttributeHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
    else
    {
        // User Information received
        UserAttributes *userInfo = (UserAttributes*)result.resultData;
        NSLog(@"User lookup attributes Received : %@", userInfo);

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"User Attributes"
            message:[userInfo description] delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
        [alert show];
    }
}
```

See Also

getUserInformation:

UserAttributes

Class: CentrifySDKError

Returns an error or exception for a method in this API.

Syntax

```
@interface CentrifySDKError : NSObject {
   @private
      NSString *_domain;
      CentrifySDKErrorCode _code;
      NSString *_message;
      NSDictionary *_userInfo;
}
```

Methods

The `CentrifySDKError` class provides the following methods:

Method	Description
code	Returns the error code.
domain	Returns the error domain.
localizedDescription	Returns the primary message for the error in a form presentable to the user.
userInfo	Returns optional additional information about the error.
`class`	Inherited from `NSObject`.
`conformsToProtocol:`	Inherited from `NSObject`.
`description`	Inherited from `NSObject`.
`debugDescription`	Inherited from `NSObject`.
`hash`	Inherited from `NSObject`.
`isEqual`	Inherited from `NSObject`.
`isKindOfClass`	Inherited from `NSObject`.
`isMemberOfClass`	Inherited from `NSObject`.
`isProxy:`	Inherited from `NSObject`.
`performSelector:`	Inherited from `NSObject`.
`performSelector:` `withObject:`	Inherited from `NSObject`.

`performSelector:`
`withObject:`
`withObject:`
	Inherited from `NSObject`.

`respondsToSelector:`	Inherited from `NSObject`.
`self:`	Inherited from `NSObject`.
`superclass:`	Inherited from `NSObject`.

Discussion

An instance of the `CentrifySDKError` class represents a single error from a call to one of the methods of this API. Subsequent calls return different error objects.

The `CentrifySDKError` object is returned through the argument of the handler callback method.

See Also

CentrifySDKResult

Method: code

Returns the error code.

Class

`CentrifySDKError`

Syntax

```
- (CentrifySDKErrorCode)code
```

Return value

An error code defined by the `CentrifySDKErrorCode` enumeration:

```
typedef enum CentrifySDKErrorCode
{
    /// Operation failed because it was canceled by user.
    CentrifySDKErrorUserHasCanceled = 1,

    /// Operation failed because a network error occurred.
    CentrifySDKErrorNetworkError,

    /// Centrify API returned an error for this opeation.
    CentrifySDKErrorAPIError
}CentrifySDKErrorCode;
```

Method: domain

Returns the error domain.

Class

`CentrifySDKError`

Syntax

```
- (NSString *)domain
```

Return value

The domain of the set of iOS MAS SDK errors:

```
extern NSString *const CentrifySDKErrorCodeDomain;
```

Discussion

The error domain is used to ensure uniqueness of error codes by differentiating between different groups of codes. All error codes for the iOS MAS API use the same domain string: `CentrifySDKErrorCodeDomain`.

Method: localizedDescription

Returns the primary message for the error in a form presentable to the user.

Class

`CentrifySDKError`

Syntax

```
- (NSString *)domain
```

Return value

A string that concatenates the domain, error code, and error message.

Discussion

You can present this string to the user.

Example

The following code sample illustrates the use of the `localizedDescription` method:

```
...
- (void) lookupUserAttributeHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
```

Method: userInfo

Returns optional additional information about the error.

Class

`CentrifySDKError`

Syntax

```
- (NSDictionary *)userInfo
```

Return value

A key-value dictionary that contains information about the error; or `nil`.

Class: CentrifySecurityToken

Provides access to the information in a parsed SAML response.

Syntax

```
@interface CentrifySecurityToken : NSObject
```

Properties

The `CentrifySecurityToken` class provides the following properties:

Property	Description
`attributes`	The set of attributes in the `CentrifySecurityToken` object.
`audience`	The address to which the response has been sent.
`authenticationMethod`	The authentication method used for the SAML response.
`expiredTime`	The time at which the SAML response will have expired.
`issuer`	The issuer of the SAML response.
`nameFormat`	The format of the subject name.
`recipient`	The address of the intended recipient of the SAML response.
`signatureType`	The type of signature used for the SAML response.
`subjectConfirmationMethod`	The subject confirmation method specified in the SAML response.
`subjectName`	The subject name in the SAML response.
`version`	The version number of the SAML response.
`xmlData`	The raw XML data of the SAML response.
`class`	Inherited from `NSObject`.
`conformsToProtocol:`	Inherited from `NSObject`.
`description`	Inherited from `NSObject`.
`debugDescription`	Inherited from `NSObject`.
`hash`	Inherited from `NSObject`.
`isEqual`	Inherited from `NSObject`.
`isKindOfClass`	Inherited from `NSObject`.
`isMemberOfClass`	Inherited from `NSObject`.
`isProxy:`	Inherited from `NSObject`.
`performSelector:`	Inherited from `NSObject`.
`performSelector:` `withObject:`	Inherited from `NSObject`.

`performSelector:`
`withObject:`
`withObject:`
	Inherited from `NSObject`.

`respondsToSelector:`	Inherited from `NSObject`.
`self:`	Inherited from `NSObject`.
`superclass:`	Inherited from `NSObject`.

Description

The `EnterpriseAuthentication.parseSecurityToken` method parses all the information in a SAML response and returns a `CentrifySecurityToken` object. You can use the methods in this class to get and set the values in this object.

See Also

getSecurityTokenForTarget: alwaysUseFreshToken: completionHandler:

parseSecurityToken:

Property: attributes

The set of attributes in the `CentrifySecurityToken` object.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, retain) NSDictionary *attributes;
```

Property value

The set of key-value pairs of all the attributes in the SAML response.

Discussion

The SAML response contains the attributes that are specified by an application-specific SAML script run by Centrify Cloud Service when the user opens the single-sign-on SaaS or mobile application. The script is created by the administrator when configuring the Generic SAML template in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

See Also

userLookupForAdditionalAttributes: completionHandler:

Property: audience

The address to which the response has been sent.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, retain) NSString *audience;
```

Property value

The URI of the audience to which the response is addressed. The audience is in the SAML assertion script configured in Centrify Cloud Manager.

Discussion

If the audience is not the current application, you should forward the SAML response to the URI specified as the audience.

The SAML assertion script is created by the administrator when configuring the Generic SAML template in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

Property: authenticationMethod

The authentication method used for the SAML response.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, retain) NSString *authenticationMedthod;
```

Property value

The authentication method as recorded in the authentication context of the SAML response.

Property: expiredTime

The time at which the SAML response will have expired.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, retain) NSDate *expiredTime;
```

Property value

The GMT time at which the SAML response will have expired.

Property: issuer

The issuer of the SAML response.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, retain) NSString *issuer;
```

Property value

The issuer name from the SAML response.

Discussion

The SAML response includes an `<Issuer>` element, which identifies the entity that issued the response. This method returns the name of the issuer from the `<Issuer>` element. You set the issuer in the Issuer field of the Application Settings tab when configuring the Generic SAML template for the application in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

Property: nameFormat

The format of the subject name.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, retain) NSString *nameFormat;
```

Property value

The format used for the subject name.

Discussion

For name formats, see *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*; OASIS Standard, 15 March 2005; at http://docs.oasis-open.org/security/saml/v2.0/.

See Also

subjectName

Property: recipient

The address of the intended recipient of the SAML response.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, retain) NSString *recipient;
```

Property value

The URI of the recipient to which the response is addressed.

Discussion

If the recipient is not the current address, you should forward the SAML response to the URI specified as the recipient. You set the recipient in the SAML assertion script in the Advanced tab of the Generic SAML template for the application in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

Property: signatureType

The type of signature used for the SAML response.

Class

```
CentrifySecurityToken
```

Syntax

```
@property (nonatomic, retain) NSString *signatureType;
```

Property value

The type of digital signature used. Possible values are `'Response'` or `'Assertion'`.

Discussion

SAML responses are digitally signed to ensure their integrity. The signature type specifies whether the SAML assertion itself or the response is signed. You set the signature type in the SAML assertion script in the Advanced tab of the Generic SAML template for the application in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

Property: subjectConfirmationMethod

The subject confirmation method specified in the SAML response.

Class

```
CentrifySecurityToken
```

Syntax

```
@property (nonatomic, retain) NSString *subjectConfirmationMethod;
```

Property value

The method used to confirm the subject.

Discussion

The subject confirmation method is a URI reference that identifies the method used to confirm the subject. SAML-defined confirmation methods are defined in *Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0*; OASIS Standard, 15 March 2005; at http://docs.oasis-open.org/security/saml/v2.0/.

Property: subjectName

The subject name in the SAML response.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, retain) NSString *subjectName;
```

Property value

The name of the subject.

Discussion

The subject name specifies the principal that is the subject of the response. This is typically the Active Directory login name of the user that is being authenticated by this SAML response. You set the subject name in the SAML assertion script in the Advanced tab of the Generic SAML template for the application in Centrify Cloud Manager. See Centrify Application Configuration Help for instructions on using the Generic SAML template.

See Also

nameFormat

Property: version

The version number of the SAML response.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, assign) NSInteger version;
```

Property value

The major and minor version of the SAML response in the form *major.minor*; for example, `1.2`.

Property: xmlData

The raw XML data of the SAML response.

Class

`CentrifySecurityToken`

Syntax

```
@property (nonatomic, retain) NSData *xmlData;
```

Property value

The SAML response encoded in Base64-encoded XML format.

See Also

getSecurityTokenForTarget: alwaysUseFreshToken: completionHandler:

Class: CentrifySDKResult

Encapsulates the result of a call to an API method, including the data and any error returned by the call.

Syntax

```
@interface CentrifySDKResult : NSObject {
   @private
      NSObject *_resultData;
      CentrifySDKError *_error;argument
}
```

Methods

The `CentrifySDKResult` class provides the following methods:

Method	Description
`error`	Returns the error object.
`resultData`	Returns the data requested by an API call.
`class`	Inherited from `NSObject`.
`conformsToProtocol:`	Inherited from `NSObject`.
`description`	Inherited from `NSObject`.

Method	Description
`debugDescription`	Inherited from `NSObject`.
`hash`	Inherited from `NSObject`.
`isEqual`	Inherited from `NSObject`.
`isKindOfClass`	Inherited from `NSObject`.
`isMemberOfClass`	Inherited from `NSObject`.
`isProxy:`	Inherited from `NSObject`.
`performSelector:`	Inherited from `NSObject`.
`performSelector:` `withObject:`	Inherited from `NSObject`.

`performSelector:`
`withObject:`
`withObject:`
	Inherited from `NSObject`.

`respondsToSelector:`	Inherited from `NSObject`.
`self:`	Inherited from `NSObject`.
`superclass:`	Inherited from `NSObject`.

Discussion

Several methods of the `EnterpriseAuthentication` class return a `CentrifySDKResult` object, either directly or as an argument to their result handler method. The `CentrifySDKResult` object contains the data returned by the method, plus a `CentrifySDKError` object. The data is valid only if the error object is `nil`.

See Also

EnterpriseAuthentication

CentrifySDKError

Method: error

Returns the error object.

Class

`CentrifySDKResult`

Syntax

```
- (CentrifySDKError *)error
```

Return value

The `CentrifySDKError` object encapsulated in the `CentrifySDKResult` object; or `nil`.

Discussion

The data in the `CentrifySDKResult` object is valid only if the error object is `nil`.

Example

The following code sample illustrates the use of the `error` method:

```
...
- (void) lookupUserAttributeHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
```

Method: resultData

Returns the data requested by an API call.

Class

`CentrifySDKResult`

Syntax

```
- (NSObject *)resultData
```

Return value

The data object encapsulated in the `CentrifySDKResult` object. The type of object depends on the method that requested it.

Discussion

The data in the `CentrifySDKResult` object is valid only if the error object is `nil`.

Example

The following code sample illustrates the use of the `resultData` method:

```
...
    {
        // User Information received
        UserInformation *userInfo = (UserInformation*)result.resultData;
        NSLog(@"User Information Received : %@", userInfo);

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"UserName"
            message:[userInfo description] delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
        [alert show];
    }
```

Class: UserAttributes

Holds a user's Active Directory attributes such as name and address.

Syntax

```
@interface UserAttributes : NSObject
```

Properties

The `UserAttributes` class provides the following properties:

Property	Description
`additionalAttributeMap`	Attributes additional to the default set.
`city`	The user's `City` attribute.
`country`	The user's `Country` attribute.
`description`	The user's `Description` attribute.
`displayName`	The user's `DisplayName` attribute.
`email`	The user's `Email` attribute.
`firstName`	The user's `FirstName` attribute.
`groups`	The groups to which the user belongs.
`initials`	The user's `Initials` attribute.
`lastName`	The user's `LastName` attribute.
`postalCode`	The user's `PostalCode` attribute.
`state`	The user's `State` attribute.
`streetAddress`	The user's `StreetAddress` attribute.
`telephone`	The user's `Telephone` attribute.
`class`	Inherited from `NSObject`.
`conformsToProtocol:`	Inherited from `NSObject`.
`description`	Inherited from `NSObject`.
`debugDescription`	Inherited from `NSObject`.
`hash`	Inherited from `NSObject`.
`isEqual`	Inherited from `NSObject`.
`isKindOfClass`	Inherited from `NSObject`.
`isMemberOfClass`	Inherited from `NSObject`.
`isProxy:`	Inherited from `NSObject`.
`performSelector:`	Inherited from `NSObject`.
`performSelector: withObject:`	Inherited from `NSObject`.

Property	Description
`performSelector: withObject: withObject:`	Inherited from `NSObject`.
`respondsToSelector:`	Inherited from `NSObject`.
`self:`	Inherited from `NSObject`.
`superclass:`	Inherited from `NSObject`.

Discussion

The `userLookupForAdditionalAttributes:completionHandler:` method returns a `userAttributes` object containing the values of Active Directory attributes for the authenticated user. The properties of this object contain the attribute values.

See Also

userLookupForAdditionalAttributes: completionHandler:

Property: additionalAttributeMap

Attributes additional to the default set.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSDictionary *additionalAtrributeMap
```

Property value

The user's additional attributes as a map of attribute names and values.

Discussion

Additional attributes, if any, are added to the `UserAttributes` object by the `userLookupForAdditionalAttributes:completionHandler:` method.

Property: city

The user's `City` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *city
```

Property value

The user's `City` attribute.

Property: country

The user's `Country` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *country
```

Property value

The user's `Country` attribute.

Method: description

The user's `Description` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *description
```

Property value

The user's `Description` attribute.

Property: displayName

The user's `DisplayName` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *displayName
```

Property value

The user's `DisplayName` attribute.

Property: email

The user's `Email` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *email
```

Property value

The user's `Email` attribute.

Property: firstName

The user's `FirstName` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *firstName
```

Property value

The user's `FirstName` attribute.

Property: groups

The groups to which the user belongs.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSArray *groups
```

Property value

An array of groups to which the user belongs.

Property: initials

The user's `Initials` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *initials
```

Property value

The user's `Initials` attribute.

Property: lastName

The user's `LastName` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *lastName
```

Property value

The user's `LastName` attribute.

Property: postalCode

The user's `PostalCode` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *postalCode
```

Property value

The user's `PostalCode` attribute.

Property: state

The user's `State` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *state
```

Property value

The user's `State` attribute.

Property: streetAddress

The user's `StreetAddress` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *streetAddress
```

Property value

The user's `StreetAddress` attribute.

Property: telephone

The user's `Telephone` attribute.

Class

`UserAttributes`

Syntax

```
@property (nonatomic, retain) NSString *telephone
```

Property value

The user's `Telephone` attribute.

Class: UserInformation

Holds a user's authentication information.

Syntax

```
@interface UserInformation : NSObject
```

Properties

The `UserInformation` class provides the following properties:

Properties	Description
`authType`	The type of authentication being used.
`lastAuthTime`	The time of the most recent authentication.

Properties	Description
`userName`	The user name used for authentication.
`class`	Inherited from `NSObject`.
`conformsToProtocol:`	Inherited from `NSObject`.
`description`	Inherited from `NSObject`.
`debugDescription`	Inherited from `NSObject`.
`hash`	Inherited from `NSObject`.
`isEqual`	Inherited from `NSObject`.
`isKindOfClass`	Inherited from `NSObject`.
`isMemberOfClass`	Inherited from `NSObject`.
`isProxy:`	Inherited from `NSObject`.
`performSelector:`	Inherited from `NSObject`.
`performSelector:` `withObject:`	Inherited from `NSObject`.

`performSelector:`
`withObject:`
`withObject:`
	Inherited from `NSObject`.

`respondsToSelector:`	Inherited from `NSObject`.
`self:`	Inherited from `NSObject`.
`superclass:`	Inherited from `NSObject`.

The `getUserInformation:` method returns a `userInformation` object containing the values of authentication attributes for the authenticated user from Centrify Cloud Service. The properties of this object contain the authentication attribute values.

See Also

getUserInformation:

UserAttributes

Property: authType

The type of authentication being used.

Class

`UserInformation`

Syntax

```
@property (nonatomic, retain) NSString *authType;
```

Property value

The authentication type.

Property: lastAuthTime

The time of the most recent authentication.

Class

`UserInformation`

Syntax

```
@property (nonatomic, retain) NSDate *lastAuthTime
```

Property value

The last time at which the user was authenticated, in milliseconds since 00:00:00 GMT, 1 January, 1970.

Property: userName

The user name used for authentication.

Class

`UserInformation`

Syntax

```
@property (nonatomic, retain) NSString *userName;
```

Property value

The user name of the authenticated user.

iOS MAS Sample Code

As illustrated in Figure 1. Centrify cloud components, there are four main components involved in Centrify Cloud Service:

- A corporate or organization's network running the Centrify Cloud Proxy Server and Active Directory.
- Centrify cloud authentication service.
- One or more providers of Internet (that is, cloud) based software (software as a service, or SaaS) including web applications that may be on-premises or hosted remotely.
- Mobile or desktop devices that need to access the web applications, where the users of the devices belong to groups and roles set up in Active Directory.

Centrify Cloud Service provides authentication and single sign-on (SSO) services for the mobile devices and for the SaaS application.

This chapter walks through the Objective-C code for a sample mobile application that uses the iOS MAS API.

If you are new to iOS programming, note that much of the UI code is handled by the Apple, Inc. integrated programming environment, Xcode, in interface files called nib files. Xcode provides a graphical interface to program these files and make connections between your view controllers and your views. See the iOS and Xcode documentation for details.

main.m

The main file of the CentrifySDKSample app performs the following actions:

1 Imports the UI kit and the App delegate header file. Step 1

2 Creates the application object and initializes the event cycle. Step 2

```
/*
Step 1
*/
#import <UIKit/UIKit.h>

#import "AppDelegate.h"
/*
Step 2
Called when the activity is first created.
*/
int main(int argc, char *argv[])
{
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    return retVal;
}
```

AppDelegate.m

The `AppDelegate.m` file of the CentrifySDKSample app implements the application delegate. The application delegate handles certain calls from the system. The code, shown below, performs the following actions:

1. Imports Java libraries. Step 1
2. Makes the app window key and visible when the app finishes launching. Step 2
3. Handles requests to open resources identified by URL. Step 3
4. Handles a request for a security token by callling the `getSecurityTokenForTarget:` method, which returns the security token for the `sdkdemo` application defined in Cloud Manager. Step 4

```
/*
Step 1
*/
#import "AppDelegate.h"
#import <CentrifySDK/EnterpriseAuthentication.h>

@interface AppDelegate(Private)

- (void)testSDKApis;

@end

@implementation AppDelegate

@synthesize window = _window;
@synthesize appNavigationController = _appNavigationController;
@synthesize homeViewController = _homeViewController;

/*
Step 2
*/
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
    (NSDictionary *)launchOptions {

    _appNavigationController.view.frame = _window.frame;
    [self.window makeKeyAndVisible];

    return YES;
}
/*
Step 3
*/
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url
    sourceApplication:(NSString *)sourceApplication annotation:(id)annotation
{
    BOOL handled = [EnterpriseAuthentication application:application openURL:url
        sourceApplication:sourceApplication annotation:annotation];
    if (handled)
    {
        return handled;
    }

    return NO;
}

@end
```

```
@implementation AppDelegate(Private)
/*
Step 4
*/
- (void)testSDKApis
{
    [EnterpriseAuthentication getSecurityTokenForTarget:@"sdkdemo"
        alwaysUseFreshToken:YES completionHandler:NULL];
}

@end
```

HomeViewController.m

The `HomeViewController.m` file of the CentrifySDKSample app implements the buttons that exercise the iOS SDK. The code, shown below, performs the following actions:

1. Loads and initializes the view. Step 1
2. Implements the **Get Centrify SAMLToken** button by calling the `getSecurityTokenForTarget:` method (see "getSecurityTokenForTarget: alwaysUseFreshToken: completionHandler:" on page 107). Step 2

   The completion handler for this method is defined in Step 9.
3. Implements the **Get User Information** button by calling the `getUserInformation:` method (see "getUserInformation:" on page 109). Step 3

   The completion handler for this method is defined in Step 10.
4. Implements the **Look Up User Attributes** button by calling the `userLookupForAdditionalAttributes:` method (see "userLookupForAdditionalAttributes: completionHandler:" on page 113). Step 4

   The completion handler for this method is defined in Step 11.
5. Implements the **Log Out** button by calling the `logout:` method (see "logout:" on page 111). Step 5

   The completion handler for this method is defined in Step 12.
6. Defines the behavior of the dialog box that appears when the user clicks the **Get Centrify SAMLToken** button. As defined in the completion handler (Step 9) for the `getSecurityTokenForTarget:` method, this dialog has three buttons: **OK**, **Parse Token**, and **Open Url**. Indices for these buttons start with 0. Step 6
7. If the user clicks the **Parse Token** button (`buttonIndex == 1`), the `switch` statement calls the `EnterpriseAuthentication parseSecurityToken:` method (see "parseSecurityToken:" on page 112) to parse the security token and displays a dialog with the title **Parsed Token** and the `description` property from the `CentrifySecurityToken` object (see "CentrifySecurityToken" on page 117). Step 7

8 If the user clicks the **Open Url** button (`buttonIndex == 2`), the `switch` statement calls the `EnterpriseAuthentication` `parseSecurityToken:` method to parse the security token, opens the URL specified in the `audience` property of the `CentrifySecurityToken` object, and sends the token data to the URL. Step 8

9 Handles the **Get Centrify SAMLToken** button by extracting the data from the token, writing the data to `NSLog`, then displaying a dialog box with the token data and the buttons **OK**, **Parse Token**, and **Open URL** Step 9

10 Handles the **Get User Information** button by extracting the user information from the token, writing the data to `NSLog`, and displaying the user name in a dialog box. Step 10

11 Handles the **Look Up User Attributes** button by extracting the user attributes from the token, writing the data to `NSLog`, and displaying the `description` attribute in a dialog box. Step 11

12 Handles the **Log Out** button by displaying a dialog that says "Logout done". Step 12

13 When called in Step 8 as part of constructing a URL, replaces unsafe characters in the data with characters that are safe for URLs. Step 13

```
#import "HomeViewController.h"
#import "WebViewController.h"
#import <CentrifySDK/EnterpriseAuthentication.h>

typedef enum alertViewTag
{
    AlertViewTagSecurityToken = 1,
    AlertViewTagUserInfformation,
    AlertViewTagUserLookup,
    AlertViewTagParsedToken,
    AlertViewTagLogout
}alertViewTag;

@interface HomeViewController (Private)

-(NSString *)encodeQuery:(NSString *)str;
- (void) getSecurityTokenHandler:(CentrifySDKResult *)result;
- (void) getUserInformationHandler:(CentrifySDKResult *)result;
- (void) lookupUserAttributeHandler:(CentrifySDKResult *)result;
- (void) logoutHandler:(CentrifySDKResult *)result;

@end

@implementation HomeViewController

@synthesize accessToken;
/*
Step 1
*/
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}
```

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}
/*
Step 2
*/
- (IBAction)getAccessToken:(id)sender
{
    self.accessToken = nil;
    [EnterpriseAuthentication getSecurityTokenForTarget:@"sdkdemo" alwaysUseFreshToken:NO
            completionHandler:^(CentrifySDKResult *result) {
        [self getSecurityTokenHandler:result];
    }];
}
/*
Step 3
*/
- (IBAction)getUserInformation:(id)sender
{
    [EnterpriseAuthentication getUserInformation:^(CentrifySDKResult *result) {
        [self getUserInformationHandler:result];
    }];
}
/*
Step 4
*/
- (IBAction)lookupUserAttribute:(id)sender
{
    NSArray *attributes = [NSArray arrayWithObjects:@"postOfficeBox", @"wWWHomePage",
nil];
    [EnterpriseAuthentication userLookupForAdditionalAttributes:attributes
completionHandler:^(CentrifySDKResult *result) {
        [self lookupUserAttributeHandler:result];
    }];
}
/*
Step 5
*/
- (IBAction)logoutUser:(id)sender{
    [EnterpriseAuthentication logout:^(CentrifySDKResult *result) {
        [self logoutHandler:result];
    }];
}

#pragma mark UIAlertViewDelegate
/*
Step 6
*/
- (void)alertView:(UIAlertView *)alertView
didDismissWithButtonIndex:(NSInteger)buttonIndex
{
    switch (alertView.tag)
    {
/*
Step 7
*/
        case AlertViewTagSecurityToken:
```

```
            if (buttonIndex == 1)
            {
                CentrifySDKResult *result = [EnterpriseAuthentication
                    parseSecurityToken:self.accessToken];
                if (result.error)
                {
                    NSLog(@"Error occurred : %@", [result.error localizedDescription]);
                    UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
                        message:[result.error localizedDescription] delegate:nil
                        cancelButtonTitle:@"OK" otherButtonTitles:nil];
                    [alert show];
                }
                else
                {
                    CentrifySecurityToken *parsedToken =
                        (CentrifySecurityToken*)result.resultData;
                    NSLog(@"Parsed Token : %@", [parsedToken description]);
                    UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"Parsed
                        Token" message:[parsedToken description] delegate:nil
                        cancelButtonTitle:@"OK" otherButtonTitles:nil];
                    [alert show];
                }
            }
/*
Step 8
*/
            else if (buttonIndex == 2)
            {
                CentrifySDKResult *result = [EnterpriseAuthentication
                    parseSecurityToken:self.accessToken];
                if (result.error)
                {
                    NSLog(@"Error occurred : %@", [result.error localizedDescription]);
                    UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
                        message:[result.error localizedDescription] delegate:nil
                        cancelButtonTitle:@"OK" otherButtonTitles:nil];
                    [alert show];
                }
                else
                {
                    CentrifySecurityToken *parsedToken =
                        (CentrifySecurityToken*)result.resultData;

// We have SAML token. Connect to your server and open a url
                    NSString *appUrl = [NSString stringWithFormat:@"%@?SAMLResponse=%@",
                        [parsedToken audience], [self encodeQuery:self.accessToken]];

                    [[UIApplication sharedApplication] openURL:[NSURL
                        URLWithString:appUrl]];
                }

            }
            break;
        default:
            break;
    }
}

@end


@implementation HomeViewController (Private)
/*
Step 9
*/
```

```
- (void) getSecurityTokenHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
    else
    {
        // auth token received
        NSString *token = (NSString*)result.resultData;
        NSLog(@"Auth Token Received : %@", token);
        self.accessToken = token;

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"Access Token"
            message:token delegate:self cancelButtonTitle:@"OK" otherButtonTitles:@"Parse
            Token", @"Open Url", nil];
        alert.tag = AlertViewTagSecurityToken;
        [alert show];
    }
}
/*
Step 10
*/
- (void) getUserInformationHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
    else
    {
        // User Information received
        UserInformation *userInfo = (UserInformation*)result.resultData;
        NSLog(@"User Information Received : %@", userInfo);

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"UserName"
            message:[userInfo description] delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
        [alert show];
    }
}
/*
Step 11
*/
- (void) lookupUserAttributeHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
    else
    {
        // User Information received
```

```
        UserAttributes *userInfo = (UserAttributes*)result.resultData;
        NSLog(@"User lookup attributes Received : %@", userInfo);

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"User Attributes"
            message:[userInfo description] delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
        [alert show];
    }
}
/*
Step 12
*/
- (void) logoutHandler:(CentrifySDKResult *)result
{
    if (result.error)
    {
        NSLog(@"Error occurred : %@", [result.error localizedDescription]);
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"SDK Error"
            message:[result.error localizedDescription] delegate:nil
            cancelButtonTitle:@"OK" otherButtonTitles:nil];
        [alert show];
    }
    else
    {
        // Logout is done
        NSLog(@"Logout done");

        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"Success"
            message:@"Logout done" delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil];
        [alert show];
    }
}
/*
Step 13
*/
-(NSString *)encodeQuery:(NSString *)str
{
    // Encode unsafe characters according to RFC 1738 and RFC 3986
    NSString* escapeStr = @":/?#[]@!$&'()*+,;= <>\"%{}|\\^~`";
    NSString *result = (__bridge_transfer NSString*)
        CFURLCreateStringByAddingPercentEscapes(kCFAllocatorDefault, (__bridge
        CFStringRef)str, nil, (__bridge CFStringRef)escapeStr, kCFStringEncodingUTF8);
    return result;
}

@end
```

SAML Java Reference

This chapter describes the classes, methods, and properties in the Centrify SAML SaaS API. The classes for implementing basic SAML operations are defined in the `com.centrify.cloud.saas` package and consist of:

Class	Description
`CentrifySaasException`	Encapsulates a general Centrify SaaS error or warning.
`CentrifySaasService`	A convenience class to connect to Centrify Cloud Service to get the SAML metadata for an application.
`CentrifySaasServiceException`	Encapsulates a general Centrify SaaS server error or warning.
`CentrifySamlFactory`	A factory class to create `ISamlResponseValidator` and `ISamlRequestGenerator` objects from metadata.
`InvalidConditionException`	An exception generated when a SAML Conditions element is invalid.
`InvalidSamlFormatException`	An exception generated when a SAML message does not match the schema.
`InvalidSignatureException`	An exception generated when a SAML signature element is not valid.
`SamlRequestGenerator`	A class that generates a SAML authentication request.
`SamlResponse`	A validated and parsed SAML response.
`SamlResponse.Attribute`	An attribute of a SAML response.
`SamlResponse.AttributeValue`	The value of an attribute of a SAML response.
`SamlResponseValidator`	A class that validates a SAML authentication response.
`StatusException`	An exception that indicates that the SAML response has failed validation.
`ValidationException`	A general SAML validation error.

In addition to the classes provided by the API, the API defines two interfaces, as follows:

Interface	Description
`ISamlRequestGenerator`	An interface for a class that generates a SAML authentication request.
`ISamlResponseValidator`	An interface for a class that validates a SAML authentication response.

Class: CentrifySaasException

Encapsulates a general Centrify SaaS error or warning.

Syntax

```
class CentrifySaasException
extends java.lang.Exception
```

Methods

The `CentrifySaasException` class provides the following methods:

Method	Description
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`addSuppressed`	Inherited from `java.lang.Throwable`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.
`getMessage`	Inherited from `java.lang.Throwable`.
`getStackTrace`	Inherited from `java.lang.Throwable`.
`getSuppressed`	Inherited from `java.lang.Throwable`.
`initCause`	Inherited from `java.lang.Throwable`.
`printStackTrace`	Inherited from `java.lang.Throwable`.
`setStackTrace`	Inherited from `java.lang.Throwable`.
`toString`	Inherited from `java.lang.Throwable`.

Class: CentrifySaasService

A convenience class to connect to Centrify Cloud Service to get the SAML metadata for an application.

Syntax

```
class CentrifySaasService
extends java.lang.Object
implements java.io.Closeable
```

Methods

The `CentrifySaasService` class provides the following methods:

Method	Description
`call`	Makes a generic GET or POST request to Centrify Cloud Service.
`close`	Logs out of Centrify Cloud Service and clears the URL, customer ID, user name, password, and any cookie set by the login.
`getSAMLMetadataForApp`	Returns the SAML identity provider metadata for the specified application.
`getSSOToken`	Returns the SAML response for the specified application.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

This is a convenience class that connects to and communicates with Centrify Cloud Service to get a SAML response and SAML metadata.

Constructor: CentrifySaasService

Creates a new `CentrifySaasService` object.

Syntax

```
public CentrifySaasService(
      String url
      String customerID
      String username
      char[] password
)
```

Parameters

url The Centrify Cloud Service URL; for example, `https://cloud.centrify.com/`.

customerID The customer ID used to connect to Centrify Cloud Manager.

username The user name used to connect to Centrify Cloud Manager. The user must have a role assignment of sysadmin or have application rights for Centrify Cloud Manager.

password The user's password.

Errors

The `CentrifySaasService` constructor may throw the following exception:

`IllegalArgumentException` The URL did not start with `https`.

Method: call

Makes a generic `GET` or `POST` request to Centrify Cloud Service.

Class

`CentrifySaasService`

Syntax

```
String call(
      String urlPath,
)
```

```
String call(
      String urlPath,
      JSONObject data
)
```

Parameters

urlPath The path, relative to the root url, passed to the constructor. For example: `SaaSManage/DownloadSAMLMetadataForApp?appKey=Salesforce`

data The data to be serialized and added to the `POST` request.

Return value

The response from Centrify Cloud Service.

Errors

The `call` method may throw one of the following exceptions:

`IOException` An I/O operation failed or was interrupted.

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

Discussion

You can use this method to implement a REST interface.

Method: close

Logs out of Centrify Cloud Service and clears the URL, customer ID, user name, password, and any cookie set by the login.

Class

`CentrifySaasService`

Syntax

```
void close()
```

Errors

The `close` method may throw the following exception:

`IOException` An I/O operation failed or was interrupted.

Method: getSAMLMetadataForApp

Returns the SAML identity provider metadata for the specified application.

Class

`CentrifySaasService`

Syntax

```
String getSAMLMetadataForApp(
      String appKey,
)
```

Parameter

`appKey` The name or key of the application for which you want the metadata. You can use either the application key or the name of the application for the *`appkey`* parameter. However, whereas the application key is guaranteed to be unique, the application name is not. The application key is assigned by Centrify Cloud Manager and is guaranteed to be universally unique.

You can find the Application key in the Cloud Manager window of the generic SAML template in the **Identity Provider Sign-in URL** field. For example:

```
https://devdog.centrify.com/run?appkey=2ff55fa1-cce8-49ec-ad24-
199333fa33a2&customerid=HT150
```

Return value

The identity provider (IdP) metadata for the specified application.

Errors

The `getSAMLMetadataForApp` method may throw one of the following exceptions:

`IOException` An I/O operation failed or was interrupted.

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

Discussion

This method downloads the metadata for the specified application from Cloud Manager and returns the metadata as a string that you can use as input to the constructor for a `CentrifySamlFactory` object.

The metadata includes the certificate for the application plus other data needed by `CentrifySamlFactory`.

Example

The following code sample illustrates the use of the `CentrifySaasService.getSamlMetadataForApp` method:

```
private static CentrifySamlFactory getCentrifySamlFactory(
        CentrifySaasService service, String appKey, String spAudience) throws Exception {
    System.out.println("Getting metadata");
    // The CentrifySaasService will log in automatically
    String metadata = service.getSAMLMetadataForApp(appKey);
    System.out.println("Successfully got metadata");

    // if the appKey doesn't match any existing application. The result is empty.
    if(metadata.isEmpty()) {
        throw new Exception("The metadata for appKey \"" + appKey + "\" is empty.");
    }

    String spIssuerName = "not used";

    // now passing the metadata to CentrifySamlFactory

    return new CentrifySamlFactory(metadata, spIssuerName, spAudience);
}
```

See Also

`CentrifySamlFactory`

Method: getSSOToken

Returns the SAML response for the specified application.

Class

`CentrifySaasService`

Syntax

```
String getSSOToken(
      String appKey,
)
```

Parameter

appKey The name, application ID, or application key of the application for which you want the SAML response. The application name is assigned by the administrator in the generic SAML template and is not guaranteed to be unique. The application ID is assigned by the administrator in the **Application ID** field of the **Application Settings** tab of the Generic SAML template and must be unique within an organization. The application key is assigned by Centrify Cloud Manager and is guaranteed to be universally unique.
You can find the Application key in the Cloud Manager window of the generic SAML template in the **Identity Provider Sign-in URL** field. For example:

```
https://devdog.centrify.com/run?appkey=2ff55fa1-cce8-49ec-ad24-
199333fa33a2&customerid=HT150
```

Return value

The SAML response for the specified application.

Errors

The `getSSOToken` method may throw one of the following exceptions:

`IOException` An I/O operation failed or was interrupted.

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

Discussion

The SAML response is signed with the private key corresponding to the certificate contained in the SAML metadata. The certificate contains a public key that you can use in the constructor for a `SamlResponseValidator` object. The methods of `SamlResponseValidator` process the SAML response and return a validated SAML response in the form of a `SamlResponse` object.

The method searches first for a match to the application ID, then for a match to the application name, then for a match to the application key. If the method finds duplicate application IDs or application names, the method fails.

Example

The following code sample illustrates the use of the `CentrifySaasService.getSSOToken` method:

```
CentrifySaasService service = new CentrifySaasService(url, customerId, username,
password.toCharArray());
```

```
CentrifySamlFactory factory;
String ssoToken;
try {
    factory = getCentrifySamlFactory(service, appKey, spAudience);

    System.out.println("Getting SSO token");
    // this is same as https://cloud.centrify.com/uprest/getSSOToken?appkey=appKey
    ssoToken = service.getSSOToken(appKey);

} finally {
    // after the service is closed, you can no longer use it.
    service.close();
}
```

See Also

SamlResponseValidator

SamlResponse

CentrifySaasServiceException

Class: CentrifySaasServiceException

Encapsulates a general Centrify SaaS server error or warning.

Syntax

```
class CentrifySaasServiceException
extends CentrifySaaException
```

Methods

The `CentrifySaasServiceException` class provides the following methods:

Method	Description
`getMessage`	Returns a message about the SaaS service exception.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.

Method	Description
`getStackTrace`	Inherited from `java.lang.Throwable`.
`getSuppressed`	Inherited from `java.lang.Throwable`.
`initCause`	Inherited from `java.lang.Throwable`.
`printStackTrace`	Inherited from `java.lang.Throwable`.
`setStackTrace`	Inherited from `java.lang.Throwable`.
`toString`	Inherited from `java.lang.Throwable`.

Method: getMessage

Returns a message about the SaaS service exception.

Class

`CentrifySaasServiceException`

Syntax

```
String getMessage()
```

Return value

An exception message.

Discussion

This method overrides `java.lang.Throwable.getMessage` to return a JSON error message with an error ID and stack trace.

Class: CentrifySamlFactory

A factory class to create `ISamlResponseValidator` and `ISamlRequestGenerator` objects from metadata.

Syntax

```
class CentrifySamlFactory
```

Methods

The `CentrifySamlFactory` class provides the following methods:

Method	Description
`getCertificate`	Returns the X509 certificate contained in the SAML metadata.
`getIdpPostUrl`	Returns the identity provider's sign-in URL.

Method	Description
`getRequestGenerator`	Returns an `ISamlRequestGenerator` object, which generates SAML authentication requests.
`getResponseValidator`	Returns an `ISamlResponseValidator` object, which validates SAML authentication responses.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

In order to validate a SAML response, you need the public key corresponding to the private key with which the response was signed. Rather than downloading the certificate and extracting the public key, you can download the SAML metadata and use the `CentrifySamlFactory` class to generate `ISamlResponseValidator` and `ISamlRequestGenerator` objects.

See Also

SamlRequestGenerator

SamlResponseValidator

Constructor: CentrifySamlFactory

Creates a new `CentrifySamlFactory` object.

Syntax

```
public CentrifySamlFactory(
      URL idpMetadataUrl,
      String idpIssuerName,
      String spIssuerName,
      String spAudience
)

public CentrifySamlFactory(
      URL idpMetadataUrl,
      String spIssuerName,
```

```
      String spAudience
)
public CentrifySamlFactory(
      String idpMetadataXml,
      String idpIssuerName,
      String spIssuerName,
      String spAudience
)
public CentrifySamlFactory(
      String idpMetadataXml,
      String spIssuerName,
      String spAudience
)
```

Parameters

idpMetadataUrl The SAML identity provider (IdP) metadata URL. This can be the URL of a file in the local file system containing the metadata, the URL of a jar file containing the metadata, or a web URL that does not require authentication to download the metadata.

idpMetadataXml The IdP metadata in XML format. You can use the `getSAMLMetadataForApp` method to get the metadata.

idpIssuerName The IdP issuer name, used for validating the SAML response. This name is the content of the **Issuer** field of the **Application Settings** tab for the application in Centrify Cloud Manager. This string is case sensitive. If this parameter is missing or `null`, the constructor gets the IdP issuer name from the metadata.

spIssuerName The service provider (SP) issuer name. This is used to generate the authentication request for SP-initiated single sign-on. It can be any string, but is usually a URL.

spAudience The SP audience, used for validating the SAML response. This should be the URL where the response is received. The audience is in the SAML assertion script in the **Advanced** tab for the application in Centrify Cloud Manager.

Errors

The `CentrifySamlFactory constructor` may throw one of the following exceptions:

`IOException` An I/O operation failed or was interrupted.

`InvalidSamlFormatException` The metadata is not in a valid format.

`CertificateException` The metadata does not contain a valid certificate.

Discussion

For the Centrify Cloud Service, you can download the metadata with the following procedure:

1 Opening a browser and log into Centrify Cloud Manager

2 Navigate to the application's profile page

3 Click the **Download Identity Provider SAML metadata** button.

Or, you can use the `CentrifySaasService` class, which logs into the Centrify mangement portal and downloads the metadata for you.

See Also

getSAMLMetadataForApp

Method: getCertificate

Returns the X509 certificate contained in the SAML metadata.

Class

`CentrifySamlFactory`

Syntax

```
X509Certificate getCertificate()
```

Return value

The X509 certificate.

Discussion

You can use this method to extract the certificate from the SAML metadata. This certificate contains the public key corresponding to the private key used to sign the SAML response. You can also download this certificate from the application's page in Centrify Cloud Manager.

Example

The following code sample illustrates the use of the `CentrifySamlFactory.getCertificate` method:

```
/**
X509Certificate certificate = factory.getCertificate();
try {
    //check certificate's validity.
    certificate.checkValidity();
} catch (CertificateExpiredException e) {
    // the certificate has expired.

    System.out.println("The certificate has been expired since " +
        certificate.getNotAfter());
     throw e;
} catch (CertificateNotYetValidException e) {
    // the certificate is not yet valid.

    System.out.println("The certificate is not yet valid until " +
        certificate.getNotBefore());
    throw e;
}
```

See Also

SamlResponseValidator

Method: getIdpPostUrl

Returns the identity provider's sign-in URL.

Class

`CentrifySamlFactory`

Syntax

```
String getIdpPostUrl()
```

Return value

The identity provider (IdP) URL; this is the sign-in URL in the IdP metadata given in the constructor for the `CentrifySamlFactory` object.

Discussion

The IdP URL is used to redirect an authentication request (`AuthnRequest`) to the identity provider. See the SAML documentation for more information about authentication requests.

Use the `getRequestGenerator` method to create a `SamlRequestGenerator` object that you can use to generate authentication requests.

See Also

getRequestGenerator

SamlRequestGenerator

Method: getRequestGenerator

Returns an `ISamlRequestGenerator` object, which generates SAML authentication requests.

Class

`CentrifySamlFactory`

Syntax

```
ISamlRequestGenerator getRequestGenerator()
```

Return value

An interface for an object that generates a SAML authentication request.

Discussion

When the login sequence is initiated by the user attempting to log in directly to your SaaS application from their computer, you can provide single-sign-on by responding to the login request with a SAML authentication request (`AuthnRequest`) addressed to the identity provider.

See Also

getIdpPostUrl

ISamlRequestGenerator

getResponseValidator

Method: getResponseValidator

Returns an `ISamlResponseValidator` object, which validates SAML authentication responses.

Class

`CentrifySamlFactory`

Syntax

```
ISamlResponseValidator getResponseValidator()
```

Return value

An interface for an object that validates SAML authentication responses.

Discussion

When you receive a SAML authentication response, you need to make sure the signature is valid and parse the response so you can extract any user attributes that you need in order to log in the user. You can use the `IamlResponseValidator` object returned by this method to validate the signature and parse the response.

Example

The following code sample illustrates the use of the `CentrifySamlFactory.getResponseValidator` method:

```
/**
CentrifySamlFactory factory;
String ssoToken;
try {
    factory = getCentrifySamlFactory(service, appKey, spAudience);

    System.out.println("Getting SSO token");
    ssoToken = service.getSSOToken(appKey);

}

X509Certificate certificate = factory.getCertificate();
```

```
try {
    //check certificate's validity.
    certificate.checkValidity();
} catch (CertificateExpiredException e) {
    // the certificate has expired.
    System.out.println("The certificate has been expired since " +
        certificate.getNotAfter());
    throw e;
}

ISamlResponseValidator validator = factory.getResponseValidator();
SamlResponse response = validator.processSamlResponseXml(ssoToken);
```

See Also

ISamlResponseValidator

getRequestGenerator

Class: InvalidConditionException

An exception generated when a SAML Conditions element is invalid.

Syntax

```
class InvalidConditionException
extends ValidationException
```

Methods

The `InvalidConditionException` class provides the following methods:

Method	Description
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`addSuppressed`	Inherited from `java.lang.Throwable`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.
`getMessage`	Inherited from `java.lang.Throwable`.
`getStackTrace`	Inherited from `java.lang.Throwable`.

Method	Description
getSuppressed	Inherited from java.lang.Throwable.
initCause	Inherited from java.lang.Throwable.
printStackTrace	Inherited from java.lang.Throwable.
setStackTrace	Inherited from java.lang.Throwable.
toString	Inherited from java.lang.Throwable.

Class: InvalidSamlFormatException

An exception generated when a SAML message does not match the schema.

Syntax

```
class InvalidSamlFormatException
extends CentrifySaasException
```

Methods

The InvalidSamlFormatException class provides the following methods:

Method	Description
clone	Inherited from java.lang.Object.
equals	Inherited from java.lang.Object.
finalize	Inherited from java.lang.Object.
getClass	Inherited from java.lang.Object.
hashCode	Inherited from java.lang.Object.
notify	Inherited from java.lang.Object.
notifyAll	Inherited from java.lang.Object.
wait	Inherited from java.lang.Object.
addSuppressed	Inherited from java.lang.Throwable.
fillInStackTrace	Inherited from java.lang.Throwable.
getCause	Inherited from java.lang.Throwable.
getLocalizedMessage	Inherited from java.lang.Throwable.
getMessage	Inherited from java.lang.Throwable.
getStackTrace	Inherited from java.lang.Throwable.
getSuppressed	Inherited from java.lang.Throwable.
initCause	Inherited from java.lang.Throwable.
printStackTrace	Inherited from java.lang.Throwable.

Method	Description
setStackTrace	Inherited from java.lang.Throwable.
toString	Inherited from java.lang.Throwable.

Class: InvalidSignatureException

An exception generated when a SAML signature element is not valid.

Syntax

```
class InvalidSignatureException
extends ValidationException
```

Methods

The InvalidSignatureException class provides the following methods:

Method	Description
clone	Inherited from java.lang.Object.
equals	Inherited from java.lang.Object.
finalize	Inherited from java.lang.Object.
getClass	Inherited from java.lang.Object.
hashCode	Inherited from java.lang.Object.
notify	Inherited from java.lang.Object.
notifyAll	Inherited from java.lang.Object.
wait	Inherited from java.lang.Object.
addSuppressed	Inherited from java.lang.Throwable.
fillInStackTrace	Inherited from java.lang.Throwable.
getCause	Inherited from java.lang.Throwable.
getLocalizedMessage	Inherited from java.lang.Throwable.
getMessage	Inherited from java.lang.Throwable.
getStackTrace	Inherited from java.lang.Throwable.
getSuppressed	Inherited from java.lang.Throwable.
initCause	Inherited from java.lang.Throwable.
printStackTrace	Inherited from java.lang.Throwable.
setStackTrace	Inherited from java.lang.Throwable.
toString	Inherited from java.lang.Throwable.

Discussion

Every SAML message is signed using a private key. If the signature is not valid when the message is received, the message is counterfeit or has been corrupted and cannot be trusted.

Class: SamlRequestGenerator

A class that generates a SAML authentication request.

Syntax

```
class SamlRequestGenerator
```

Implements

`ISamlRequestGenerator`

Methods

The `SamlRequestGenerator` class provides the following methods:

Method	Description
`generateRequest`	Generates a SAML authentication request in Base64-encoded XML format.
`generateRequestXml`	Generates a SAML authentication request in XML format.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

This class generates SAML authentication requests in XML or Base64-encoded XML format.

When the login sequence is initiated by the user attempting to log in directly to your SaaS application from their computer, you can still provide single-sign-on by redirecting the request to the identity provider (IdP). To do so, respond to the login request with a SAML authentication request (`AuthnRequest`) addressed to the IdP. Use the `SamlRequestGenerator` class to generate the authentication request.

You can use the `CentrifySamlFactory.getRequestGenerator` method to create an `ISamlRequestGenerator` object. You can use the `CentrifySamlFactory.getIdpPostUrl` method to obtain the redirection URL.

See Also

ISamlRequestGenerator

getRequestGenerator

getIdpPostUrl

Constructor: SamlRequestGenerator

Creates a new `SamlRequestGenerator` object.

Syntax

```
public SamlRequestGenerator(
      String spIssuerName,
)
```

Parameters

`spIssuerName` The service provider (SP) issuer name. This can be any string, but is usually a URL.

Method: generateRequest

Generates a SAML authentication request in Base64-encoded XML format.

Class

`SamlRequestGenerator`

Syntax

```
String generateRequest()
```

Return value

The authentication request in Base64-encoded XML format.

Errors

The `generateRequest` method may throw the following exception:

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

Discussion

Because the Centrify cloud server uses HTTP POST binding, you should send the SAML `AuthnRequest` message as Base64-encoded XML to Centrify cloud. You can use the `generateRequestXml` method to send XML-encoded `AuthnRequest` messages to other servers.

See Also

generateRequestXml

Method: generateRequestXml

Generates a SAML authentication request in XML format.

Class

`SamlRequestGenerator`

Syntax

```
String generateRequestXml()
```

Return value

The authentication request in XML format.

Errors

The `generateRequest` method may throw the following exception:

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

Discussion

Because the Centrify cloud server uses HTTP POST binding, you should use the `generateRequest` method to send the SAML `AuthnRequest` message as Base64-encoded XML to Centrify cloud. You can use the `generateRequestXml` method to send XML-encoded `AuthnRequest` messages to other servers.

See Also

generateRequest

Class: SamlResponse

A validated and parsed SAML response.

Syntax

```
class SamlResponse
```

Methods

The `SamlResponse` class provides the following methods:

Method	Description
`getAttribute`	Returns the specified attribute.
`getAttributes`	Returns the list of attributes in a SAML response.
`getAttributeValues`	Returns the values of the specified attribute.
`getNameFormat`	Returns the format of the response's name ID.
`getNameid`	Returns the response's name identifier.
`toString`	Returns the SAML response object as a string.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

This class represents a validated SAML response. The SAML response contains attributes that are specific to the authenticated user.

You obtain a `SamlResponse` object from the methods of the `SamlResponseValidator` class.

See Also

SamlResponse.Attribute

SamlResponse.AttributeValue

SamlResponseValidator

Method: getAttribute

Returns the specified attribute.

Class

`SamlResponse`

Syntax

```
SamlResponse.Attribute getAttribute(
```

```
      String attributeName
)
```

Parameters

attributeName The case-sensitive name of the attribute.

Return value

The attribute with the specified name, or `null` if there was no such attribute in the response.

Discussion

The SAML response contains attributes that are specific to the authenticated user.

Example

The following code sample illustrates the use of the `SamlResponse.getAttribute` method:

```
if (emailAttr != null) {
    System.out.println(" Display Email attributes");
    System.out.println(" name format = " + emailAttr.getNameFormat());
    System.out.println(" Display attributes with \"format\"");
    List<SamlResponse.AttributeValue> attrValues = emailAttr.getValues();
    for (SamlResponse.AttributeValue attrValue : attrValues) {
        System.out.println("  " + attrValue.getFormat() + ": "
            + attrValue.getValue());
    }
} else {
    System.out.println(" no Email attribute");
}
```

See Also

SamlResponse.Attribute

SamlResponse.AttributeValue

getAttributes

getAttributeValues

Method: getAttributes

Returns the list of attributes in a SAML response.

Class

`SamlResponse`

Syntax

```
List<SamlResponse.Attribute> getAttributes()
```

Return value

The list of attributes in the response.

Discussion

The SAML response contains attributes that are specific to the authenticated user.

Example

The following code sample illustrates the use of the `SamlResponse.getAttributes` method:

```
private List<SamlResponse.Attribute>
        extractAttributes(List<AttributeStatement> attributeStatements) {
    List<SamlResponse.Attribute> resAttributes =
        new ArrayList<SamlResponse.Attribute>();
    if (attributeStatements == null) {
        return resAttributes;
    }

    for (AttributeStatement attributeStatement : attributeStatements) {
        List<Attribute> attributes = attributeStatement.getAttributes();
        if (attributes == null) {
            continue;
        }
...
```

See Also

getAttribute

getAttributeValues

SamlResponse.Attribute

Method: getAttributeValues

Returns the values of the specified attribute.

Class

`SamlResponse`

Syntax

```
List<String> getAttributeValues(
      String attributeName
)
```

Parameters

`attributeName` The case-sensitive name of the attribute.

Return value

The values of the attribute with the specified name, or `null` if there was no such attribute in the response. The list is read-only.

Discussion

Use this method if you have the name of the attribute and want the attribute values as strings. If you have a `SamlResponse.Attribute` object, use the `SamlResponse.Attribute.getStringValues` method instead. To get attribute values as `AttributeValue` objects, use the `SamlResponse.Attribute.getValues` method.

The SAML response contains attributes that are specific to the authenticated user.

Example

The following code sample illustrates the use of the `SamlResponse.getAttributeValues` method:

```
private List<SamlResponse.Attribute>
        extractAttributes(List<AttributeStatement> attributeStatements) {
    List<SamlResponse.Attribute> resAttributes =
        new ArrayList<SamlResponse.Attribute>();
    if (attributeStatements == null) {
        return resAttributes;
    }

    for (AttributeStatement attributeStatement : attributeStatements) {
        List<Attribute> attributes = attributeStatement.getAttributes();
        if (attributes == null) {
            continue;
        }

        for (Attribute attribute : attributes) {
            List<SamlResponse.AttributeValue> attributeValues =
                extractAttributeValues(attribute.getAttributeValues());

            resAttributes.add(new SamlResponse.Attribute(
                attribute.getAttributeName()
                , attribute.getAttributeNamespace()
                , null
                , attributeValues));
        }
    }

     return resAttributes;
}
```

See Also

getAttributes

getStringValues

getValues

SamlResponse.Attribute

SamlResponse.AttributeValue

Method: getNameFormat

Returns the format of the response's name ID.

Class

`SamlResponse`

Syntax

```
String getNameFormat()
```

Return value

The format of the name ID, such as persistent, transient, or email address.

Example

The following code sample illustrates the use of the `SamlResponse.getNameFormat` method:

```
private static void displaySamlResponse(SamlResponse response) {

    System.out.println(" name format = " + response.getNameFormat());
...
```

See Also

getNameid

SamlResponse.Attribute.getNameFormat

Method: getNameid

Returns the response's name identifier.

Class

`SamlResponse`

Syntax

```
String getNameid()
```

Return value

The SAML name identifier used to identify a user.

Example

The following code sample illustrates the use of the `SamlResponse.getNameid` method:

```
private static void displaySamlResponse(SamlResponse response) {
System.out.println();

System.out.println("The id is \"" + response.getNameid() + "\"");
}
```

See Also

getNameFormat

Method: toString

Returns the SAML response object as a string.

Class

`SamlResponse`

Syntax

```
String toString()
```

Return value

The SAML response object as a string.

Discussion

This method overrides `java.lang.object.toString`. It returns the contents of the SAML response object as a string with labels for the components (`nameid=`, `nameFormat=`, `attributes=`).

See Also

SamlResponse.Attribute.toString

Class: SamlResponse.Attribute

An attribute of a SAML response.

Syntax

```
static class SamlResponse.Attribute
```

Methods

The `SamlResponse.Attribute` class provides the following methods:

Method	Description
`getFriendlyName`	Returns the friendly name of the attribute.
`getName`	Returns the name of the attribute.
`getNameFormat`	Returns the format of the attribute name.
`getStringValues`	Returns attribute values in string format.
`getValues`	Returns attribute values.
`toString`	Returns the attribute object as a string.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.

Method	Description
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

This class represents an attribute of a validated SAML response. The SAML response contains attributes that are specific to the authenticated user.

You obtain a `SamlResponse` object from the methods of the `SamlResponseValidator` class and you use the `SamlResponse.getAttributes` method to obtain a list of `SamlResponse.Attribute` objects.

See Also

SamlResponse

SamlResponseValidator

getAttributes

SamlResponse.AttributeValue

Method: getFriendlyName

Returns the friendly name of the attribute.

Class

`SamlResponse.Attribute`

Syntax

```
String getFriendlyName()
```

Return value

The human-readable name of the attribute; or `null` if no friendly name is defined.

Discussion

The friendly name is optional.

See Also

getName

Method: getName

Returns the name of the attribute.

Class

`SamlResponse.Attribute`

Syntax

```
String getName()
```

Return value

The canonical name of the attribute.

See Also

getFriendlyName

getNameFormat

Method: getNameFormat

Returns the format of the attribute name.

Class

`SamlResponse.Attribute`

Syntax

```
String getNameFormat()
```

Return value

The format of the attribute name; or `null` if no name format is defined.

Discussion

The name format is defined in the SAML specification (see *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*; OASIS Standard, 15 March 2005; at http://docs.oasis-open.org/security/saml/v2.0/).

The attribute name format is optional; this method may return `null`.

Example

The following code sample illustrates the use of the `SamlResponse.getAttribute.getNameFormat` method:

```
if (emailAttr != null) {
    System.out.println(" Display Email attributes");
    System.out.println(" name format = " + emailAttr.getNameFormat());
    System.out.println(" Display attributes with \"format\"");
    List<SamlResponse.AttributeValue> attrValues = emailAttr.getValues();
```

```
    for (SamlResponse.AttributeValue attrValue : attrValues) {
        System.out.println("  " + attrValue.getFormat() + ": "
            + attrValue.getValue());
    }
} else {
    System.out.println(" no Email attribute");
}
```

See Also

getName

Method: getStringValues

Returns attribute values in string format.

Class

`SamlResponse.Attribute`

Syntax

```
List<String> getStringValues()
```

Return value

The list of attribute values in string format. Returns an empty list if the attribute does not contain any values.The list is read-only.

Discussion

If you have a `SamlResponse.Attribute` object, use this method to get attribute values as strings. To get attribute values as `AttributeValue` objects, use the `getValues` method. If you have the name of an attribute, use the `getAttributeValues` method.

The SAML response contains attributes that are specific to the authenticated user.

Example

The following code sample illustrates the use of the `SamlResponse.Attribute.getStringValues` method:

```
if (emailAttr != null) {
    System.out.println(" Display Email attributes");
    System.out.println(" name format = " + emailAttr.getNameFormat());
    System.out.println(" Display attributes without \"format\"");
    List<String> attrStringValues = emailAttr.getStringValues();
    System.out.println("  " + attrStringValues);
} else {
    System.out.println(" no Email attribute");
}
```

See Also

getValues

getAttributes

getAttributeValues

SamlResponse.AttributeValue

Method: getValues

Returns attribute values.

Class

`SamlResponse.Attribute`

Syntax

```
List<SamlResponse.AttributeValue> getValues()
```

Return value

The list of attribute values. Returns an empty list if the attribute does not contain any values. The list is read-only.

Discussion

Use this method to get attribute values as `AttributeValue` objects. To get the attribute values as strings, use the `getStringValues` method. If you have the name of an attribute, use the `getAttributeValues` method.

The SAML response contains attributes that are specific to the authenticated user.

Example

The following code sample illustrates the use of the `SamlResponse.Attribute.getValues` method:

```
private static void displaySamlResponse(SamlResponse response) {
    System.out.println();

    System.out.println("The id is \"" + response.getNameid() + "\"");

    SamlResponse.Attribute emailAttr = response.getAttribute("Email");
    if (emailAttr != null) {
        System.out.println(" Display Email attributes");
        System.out.println(" name format = " + emailAttr.getNameFormat());
        System.out.println(" Display attributes with \"format\"");
        List<SamlResponse.AttributeValue> attrValues = emailAttr.getValues();
        for (SamlResponse.AttributeValue attrValue : attrValues) {
            System.out.println("  " + attrValue.getFormat() + ": "
                + attrValue.getValue());
        }
...
```

See Also

getStringValues

`getAttributeValues`

SamlResponse.AttributeValue

Method: toString

Returns the attribute object as a string.

Class

`SamlResponse.Attribute`

Syntax

```
String toString()
```

Return value

The attribute object as a string.

Discussion

This method overrides `java.lang.object.toString`. It returns the contents of the attribute object as a string with labels for the components (`name=`, `nameFormat=`, `friendlyName=`, `values=`).

See Also

SamlResponse.toString

Class: SamlResponse.AttributeValue

The value of an attribute of a SAML response.

Syntax

```
static class SamlResponse.AttributeValue
```

Methods

The `SamlResponse.AttributeValue` class provides the following methods:

Method	Description
`getFormat`	Returns the format of the attribute value.
`getValue`	Returns the attribute value.
`toString`	Returns the attribute value as a string.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.

Method	Description
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

See Also

getAttributes

Method: getFormat

Returns the format of the attribute value.

Class

`SamlResponse.AttributeValue`

Syntax

```
String getFormat()
```

Return value

The format of the attribute value, or `null` if no format is defined.

Method: getValue

Returns the attribute value.

Class

`SamlResponse.AttributeValue`

Syntax

```
String getValue()
```

Return value

The attribute value.

Discussion&*&*

Use this method if you have an `AttributeValue` object and want only the value of the attribute. For a string that includes labels, use the `toString` method. If you have the name of the attribute and want the attribute values as strings, use the

`SamlResponse.getAttributeValues` method. If you have a `SamlResponse.Attribute` object, you can use the `SamlResponse.Attribute.getStringValues` method.

See Also

toString

getAttributeValues

getStringValues

Method: toString

Returns the attribute value as a string.

Class

`SamlResponse.AttributeValue`

Syntax

```
String toString()
```

Return value

The attribute value as a string.

Discussion

This method overrides `java.lang.object.toString`. It returns the contents of the attribute value as a string with labels (`format=`, `value=`).

Use this method if you have an `AttributeValue` object and want the value of the attribute as a string with labels. For the value alone, use the `getValue` method. If you have the name of the attribute and want the attribute values as strings, use the `SamlResponse.getAttributeValues` method. If you have a `SamlResponse.Attribute` object, you can use the `SamlResponse.Attribute.getStringValues` method.

See Also

getValue

getAttributeValues

getStringValues

Class: SamlResponseValidator

A class that validates a SAML authentication response.

Syntax

```
class SamlResponseValidator
```

Methods

The `SamlResponseValidator` class provides the following methods:

Method	Description
`processSamlResponse`	Validates a SAML authentication response that is in Base64-encoded XML format.
`processSamlResponseXml`	Validates a SAML authentication response that is in XML format.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

This class validates a SAML response that is in XML or Base64-encoded XML format. This class supports both SAML 1 and SAML 2 protocols.

This class is an implementation of the `ISamlResponseValidator` interface. You can use the `SamlFactory.getResponseValidator` method to return an `ISamlResponseValidator` object.

See Also

getResponseValidator

ISamlResponseValidator

Constructor: SamlResponseValidator

Creates a new `SamlResponseValidator` object.

Syntax

```
public SamlResponseValidator(
      PublicKey idpPublicKey,
      String idpIssuerName,
      String audience,
```

)

Parameters

idpPublicKey The public key that matches the private key used by the identity provider (IdP) to sign the SAML response.

idpIssuerName The IdP issuer name. This name is the content of the **Issuer** field of the **Application Settings** tab for the application in Centrify Cloud Manager. This string is case sensitive.

audience The URI of the audience to which the response is addressed. The audience is in the SAML assertion script in the **Advanced** tab for the application in Centrify Cloud Manager. This string is case sensitive.

Method: processSamlResponse

Validates a SAML authentication response that is in Base64-encoded XML format.

Class

```
SamlResponseValidator
```

Syntax

```
SamlResponse processSamlResponse(
      String samlResponse,
)
```

Parameters

samlResponse The Base64-encoded SAML response you want to validate.

Return value

A validated SAML response.

Errors

The `processSamlResponse` method may throw one of the following exceptions:

`InvalidSignatureException` The digital signature was not valid, indicating the response was counterfeit or corrupted.

`InvalidConditionException` A specified condition was not met.

`ValidationException` An error occurred while validating the response.

`StatusException` Validation did not succeed.

`InvalidSamlFormatException` The response does not conform to the SAML schema.

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

Discussion

This method returns a `SamlResponse` object, which you can use to get information about the name and attributes of the response. Once you have validated a response, you can trust that the user whose browser sent the response has been authenticated and is authorized to use your SaaS application.

The Centrify cloud server uses HTTP POST binding and therefore sends the SAML response as Base64-encoded XML. You can use the `processSamlResponseXml` method to process XML-encoded `Response` messages from other servers.

Example

The following code sample illustrates the use of the `SamlResponseValidator.processSamlResponse` method:

```
public static void main(String[] args) throws Exception {
    if (args == null || args.length != 1) {
        System.out.println("Usage: <file path to Base64 saml response>");
        return;
    }

    URL idpMetadataUrl = new File("idp.xml").toURI().toURL();

    // case-sensitive. This value must match the Issuer element
    String idpIssuerName = "Centrify";

    // case-sensitive. This value must match the Audience element
    String spAudience = "https://sp.centrify.com/login";

    // This is the Service Provider Issuer name.
    // This is not used to create ISamlResponseValidator.
    String spIssuerName = "spIssuerName";

    //create a factory object
    // The factory can create ISamlResponseValidator and ISamlRequestGenerator.
    CentrifySamlFactory factory = new CentrifySamlFactory(
        idpMetadataUrl,
        idpIssuerName,
        spIssuerName,
        spAudience
    );

    X509Certificate certificate = factory.getCertificate();
    try {
        //check certificate's validity.
        certificate.checkValidity();
    } catch (CertificateExpiredException e) {
        // the certificate has expired.

        System.out.println("The certificate has been expired since "
            + certificate.getNotAfter());
        throw e;
        } catch (CertificateNotYetValidException e) {
            // the certificate is not yet valid.

            System.out.println("The certificate is not valid until "
                + certificate.getNotBefore());
            throw e;
        }
```

```
        // get the ISamlResponseValidator from the factory.
        ISamlResponseValidator validator = factory.getResponseValidator();

        // load the Base64 SAML response from a file.
        String base64input = getBase64SamlResponse(args[0]);

        // validate and parse the sample response.
        // if the input is in XML format, call
        // processSamlResponseXML(String) instead
        SamlResponse response = validator.processSamlResponse(base64input);

        //If no exception is thrown, the validation is success.
        System.out.println("The SAML response is valid.");
        System.out.println(response);

        displaySamlResponse(response);
    }
}
```

See Also

SamlResponse

processSamlResponseXml

Method: processSamlResponseXml

Validates a SAML authentication response that is in XML format.

Class

`SamlResponseValidator`

Syntax

```
SamlResponse processSamlResponseXml(
      String samlResponseXML,
)
```

Parameters

`samlResponse` The XML-encoded SAML response you want to validate.

Return value

A validated SAML response.

Errors

The `processSamlResponseXml` method may throw one of the following exceptions:

`InvalidSignatureException` The digital signature was not valid, indicating the response was counterfeit or corrupted.

`InvalidConditionException` A specified condition was not met.

`ValidationException` An error occurred while validating the response.

`StatusException` Validation did not succeed.

`InvalidSamlFormatException` The response does not conform to the SAML schema.

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

Discussion

This method returns a `SamlResponse` object, which you can use to get information about the name and attributes of the response. Once you have validated a response, you can trust that the user whose browser sent the response has been authenticated and is authorized to use your SaaS application.

The Centrify cloud server uses HTTP POST binding and therefore sends the SAML response as Base64-encoded XML. You can use this `processSamlResponseXml` method to process XML-encoded `Response` messages from other servers.

See Also

processSamlResponse

SamlResponse

Class: StatusException

An exception that indicates that the SAML response has failed validation.

Syntax

```
class StatusException
extends CentrifySaasException
```

Methods

The `StatusException` class provides the following methods:

Method	Description
`getDetail`	Returns information about the status when an exception is thrown.
`getMessage`	Returns a message about the exception.
`getStatusCode`	Returns a status code for the exception.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.

Method	Description
`wait`	Inherited from `java.lang.Object`.
`addSuppressed`	Inherited from `java.lang.Throwable`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.
`getStackTrace`	Inherited from `java.lang.Throwable`.
`getSuppressed`	Inherited from `java.lang.Throwable`.
`initCause`	Inherited from `java.lang.Throwable`.
`printStackTrace`	Inherited from `java.lang.Throwable`.
`setStackTrace`	Inherited from `java.lang.Throwable`.
`toString`	Inherited from `java.lang.Throwable`.

Discussion

SAML status codes, details, and messages are described in *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*; OASIS Standard, 15 March 2005; at http://docs.oasis-open.org/security/saml/v2.0/

Method: getDetail

Returns information about the status when an exception is thrown.

Class

```
StatusException
```

Syntax

```
String getDetail()
```

Return value

Detailed status information, or `null` if there is no detail.

Discussion

When the status is not `"success"`, the `getDetail` message may provide more information about the status.

See Also

CentrifySaasException

Method: getMessage

Returns a message about the exception.

Class

`StatusException`

Syntax

```
String getMessage()
```

Return value

An exception message, or a `null` string if there is no message.

Discussion

This method overrides `java.lang.Throwable.getMessage` to return a SAML status message.

Method: getStatusCode

Returns a status code for the exception.

Class

`StatusException`

Syntax

```
String getStatusCode()
```

Return value

A status code.

Discussion

SAML status codes are listed and described in *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*; OASIS Standard, 15 March 2005; at http://docs.oasis-open.org/security/saml/v2.0/

Class: ValidationException

A general SAML validation error.

Syntax

```
class ValidationException
extends CentrifySaasException
```

Methods

The `ValidationException` class provides the following methods:

Method	Description
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`addSuppressed`	Inherited from `java.lang.Throwable`.
`fillInStackTrace`	Inherited from `java.lang.Throwable`.
`getCause`	Inherited from `java.lang.Throwable`.
`getLocalizedMessage`	Inherited from `java.lang.Throwable`.
`getMessage`	Inherited from `java.lang.Throwable`.
`getStackTrace`	Inherited from `java.lang.Throwable`.
`getSuppressed`	Inherited from `java.lang.Throwable`.
`initCause`	Inherited from `java.lang.Throwable`.
`printStackTrace`	Inherited from `java.lang.Throwable`.
`setStackTrace`	Inherited from `java.lang.Throwable`.
`toString`	Inherited from `java.lang.Throwable`.

Interface: ISamlRequestGenerator

An interface for a class that generates a SAML authentication request.

Syntax

```
interface ISamlRequestGenerator
```

Implemented by

`SamlRequestGenerator`

Methods

The `ISamlRequestGenerator` interface defines the following methods:

Method	Description
`generateRequest`	Generates a SAML authentication request in Base64-encoded XML format.
`generateRequestXml`	Generates a SAML authentication request in XML format.

Discussion

This interface generates SAML authentication requests in XML or Base64-encoded XML format.

When the login sequence is initiated by the user attempting to log in directly to your SaaS application from their computer, you can still provide single-sign-on by redirecting the request to the identity provider (IdP). To do so, respond to the login request with a SAML authentication request (`AuthnRequest`) addressed to the IdP. Use the `SamlRequestGenerator` class to generate the authentication request.

You can use the `CentrifySamlFactory.getRequestGenerator` method to create an `ISamlRequestGenerator` object. You can use the `CentrifySamlFactory.getIdpPostUrl` method to obtain the redirection URL.

See Also

SamlRequestGenerator

ISamlResponseValidator

Method: generateRequest

Generates a SAML authentication request in Base64-encoded XML format.

Interface

`ISamlRequestGenerator`

Syntax

```
String generateRequest()
```

Return value

The authentication request in Base64-encoded XML format.

Errors

The `generateRequest` method may throw the following exception:

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

Discussion

This interface method is fully implemented by the `SamlRequestGenerator.generateRequest` method.

See Also

ISamlRequestGenerator.generateRequestXml

SamlRequestGenerator.generateRequest

Method: generateRequestXml

Generates a SAML authentication request in XML format.

Interface

`ISamlRequestGenerator`

Syntax

```
String generateRequestXml()
```

Return value

The authentication request in XML format.

Errors

The `generateRequest` method may throw the following exception:

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

Discussion

This interface method is fully implemented by the `SamlRequestGenerator.generateRequestXml` method.

See Also

ISamlRequestGenerator.generateRequest

SamlRequestGenerator.generateRequestXml

Interface: ISamlResponseValidator

An interface for a class that validates a SAML authentication response.

Syntax

```
interface ISamlResponseValidator
```

Implemented by

`SamlResponseValidator`

Methods

The `ISamlResponseValidator` interface defines the following methods:

Method	Description
`processSamlResponse`	Validates a SAML authentication response that is in Base64-encoded XML format.
`processSamlResponseXml`	Validates a SAML authentication response that is in XML format.

Discussion

This interface validates SAML authentication responses in XML or Base64-encoded XML format. This interface supports both SAML 1 and SAML 2 protocols.

See Also

SamlResponseValidator

Method: processSamlResponse

Validates a SAML authentication response that is in Base64-encoded XML format.

Interface

`ISamlResponseValidator`

Syntax

```
SamlResponse processSamlResponse(
      String samlResponse,
)
```

Parameters

`samlResponse` The Base64-encoded SAML response you want to validate.

Return value

A validated SAML response.

Errors

The `processSamlResponse` method may throw one of the following exceptions:

`InvalidSignatureException` The digital signature was not valid, indicating the response was counterfeit or corrupted.

`InvalidConditionException` A specified condition was not met.

`ValidationException` An error occurred while validating the response.

`StatusException` Validation did not succeed.

`InvalidSamlFormatException` The response does not conform to the SAML schema.

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

`NullPointerException` The specified source is `null`.

Discussion

This interface method is fully implemented by the `SamlResponseValidator.processSamlResponse` method.

See Also

processSamlResponseXml

SamlResponseValidator.processSamlResponse

Method: processSamlResponseXml

Validates a SAML authentication response that is in XML format.

Interface

`ISamlResponseValidator`

Syntax

```
SamlResponse processSamlResponseXml(
      String samlResponseXML,
)
```

Parameters

`samlResponse` The XML-encoded SAML response you want to validate.

Return value

A validated SAML response.

Errors

The `processSamlResponseXml` method may throw one of the following exceptions:

`InvalidSignatureException` The digital signature was not valid, indicating the response was counterfeit or corrupted.

`InvalidConditionException` A specified condition was not met.

`ValidationException` An error occurred while validating the response.

`StatusException` Validation did not succeed.

`InvalidSamlFormatException` The response does not conform to the SAML schema.

`CentrifySaasException` An error occurred in the Centrify code. Catch the exception for an explanation.

`NullPointerException` The specified source is `null`.

Discussion

This interface method is fully implemented by the `SamlResponseValidator.processSamlResponseXml` method.

See Also

processSamlResponse

SamlResponseValidator.processSamlResponseXml

SAML Java Sample Code

As illustrated in Figure 1. Centrify cloud components, there are four main components involved in Centrify Cloud Service:

- A corporate or organization's network running the Centrify Cloud Proxy Server and Active Directory.
- Centrify cloud authentication service.
- One or more providers of Internet (that is, cloud) based software (software as a service, or SaaS) including web applications that may be on-premises or hosted remotely.
- Mobile or desktop devices that need to access the web applications, where the users of the devices belong to groups and roles set up in Active Directory.

Centrify Cloud Service provides authentication and single sign-on (SSO) services for the mobile devices and for the SaaS application.

This chapter walks through the Java code for sample web routines that use the SAML Java API to validate and read a SAML response. The first routine shows how to download the SAML response from the identity provider. The second routine shows how to validate and parse a SAML routine in a file, such as one received from a mobile application.

CentrifySaasServiceExample

This sample uses the `CentrifySaasService` class (see "CentrifySaasService" on page 144) to get the SAML metadata for the application from the Cloud Management Portal, get a SAML response for the application, and verify the SAML response using the information from the SAML metadata.

The sample code file `CentrifySaasServiceExample.java` performs the following actions:

1. Imports Java libraries. Step 1
2. Imports Centrify SaaS libraries. Step 2
3. Defines the `main` method. Step 3
4. If this method is called without any arguments or the wrong number of arguments, outputs a list of the expected arguments. Step 4
5. If the method is called with the right number of arguments, outputs the values of the arguments (except for the password, which is replaced by a string of asterisks). The arguments are the parameters needed as input to the constructor for a `CentrifySaasService` object (see "CentrifySaasService" on page 145).Step 5

Note In production code, to avoid security vulnerabilities, it is important to check every input argument for type, length, and unexpected characters. It is also important to avoid transmitting a password over an unencrypted channel.

6 Instantiates a `CentrifySaasService` object. You can use this object to contact the identity provider and download the SAML metadata (Step 15) and the SAML response (Step 8) for your application. Step 6

Note You use the metadata to validate the SAML response. If your SaaS application is a back end for a mobile application, you can obtain the SAML response from the mobile application. Also, rather than using the `CentrifySaasService` object, you can obtain the metadata in other ways as well, such as downloading it directly from Centrify Cloud Manager. See SamlResponseValidatorExample1 for sample code that validates a SAML response that is in a file.

7 Calls the `getCentrifySamlFactory` subroutine (Step 14) to instantiate a `CentrifySamlFactory` object. Step 7

8 Uses the `CentrifySaasService` object to obtain the SAML response as a string. Step 8

9 Closes the connection to the IdP. Step 9

10 Uses the `CentrifySamlFactory` object to get the signing certificate used by the IdP and uses a Java method to validate the certificate. Step 10

11 Uses the `CentrifySamlFactory` object to create a `ISamlResponseValidator` object. Step 11

12 Uses the SAML response string as input to the `SamlResponseValidator.processSamlResponseXml` method to validate the SAML response. If the method does not throw an exception, the SAML response is valid; that is, it has not been compromised or corrupted as indicated by the fact that its signature is still valid. Step 12

13 Calls the `displaySAMLResponse` subroutine (Step 17) to display the data in the SAML response. Step 13

14 Defines a subroutine that instantiates a `CentrifySamlFactory` object. Step 14

15 Uses the `CentrifySaasService` object to obtain the metadata needed to instantiate the `CentrifySamlFactory` object. Step 15

16 Instantiates a `CentrifySamlFactory` object. Step 16

17 Defines a subroutine to display the data returned in the SAML response. Step 17

18 Uses methods of the `SamlResponse` class (see "SamlResponse" on page 162) to get the name identifier and the Active Directory Email attribute from the SAML response. Step 18

19 Uses methods of the `SamlResponse.Attribute` subclass (see "SamlResponse.Attribute" on page 168) to get the format and values of the Email attribute. Step 19

20 Illustrates the use of the `Attribute.getStringValues` method (page 171) to get the values of the Email attribute without the format. Step 20.

21 Illustrates the use of the `SamlResponse.GetAttributeValues` method (page 165) to get the values of an attribute. Step 21

```
package com.centrify;
/*
Step 1
*/
import java.security.cert.CertificateExpiredException;
import java.security.cert.CertificateNotYetValidException;
import java.security.cert.X509Certificate;
import java.util.List;
/*
Step 2
*/
import com.centrify.cloud.saas.CentrifySaasService;
import com.centrify.cloud.saas.CentrifySamlFactory;
import com.centrify.cloud.saas.ISamlResponseValidator;
import com.centrify.cloud.saas.SamlResponse;

public class CentrifySaasServiceExample {
/*
Step 3
*/
    public static void main(String[] args) throws Exception {
/*
Step 4
*/
        if (args == null || args.length != 6) {
            System.out.println("url customerId username password appkey spAudience");
            return;
        }
/*
Step 5
*/
        String url = args[0];
        System.out.println("url: " + url);

        String customerId = args[1];
        System.out.println("customerId: " + customerId);

        String username = args[2];
        System.out.println("username: " + username);

        String password = args[3];
        System.out.println("password: ******");

        String appKey = args[4];
        System.out.println("appKey: " + appKey);

        String spAudience = args[5];
        System.out.println("spAudience: " + appKey);

/*
Step 6
*/
        CentrifySaasService service = new CentrifySaasService(url, customerId,
            username, password.toCharArray());
        CentrifySamlFactory factory;
        String ssoToken;
/*
```

```
Step 7
*/
        try {
            factory = getCentrifySamlFactory(service, appKey, spAudience);

/*
Step 8
*/
            System.out.println("Getting SSO token");
            // this is same as https://cloud.centrify.com/uprest/getSSOToken?appkey=appKey
            ssoToken = service.getSSOToken(appKey);

        } finally {
/*
Step 9
*/
            // after the service is closed, you can no longer use it.
            service.close();
        }
/*
Step 10
*/
        X509Certificate certificate = factory.getCertificate();
        try {
            //check certificate's validity.
            certificate.checkValidity();
        } catch (CertificateExpiredException e) {
            // the certificate has expired.

            System.out.println("The certificate has been expired since " +
                certificate.getNotAfter());
            throw e;
        } catch (CertificateNotYetValidException e) {
            // the certificate is not yet valid.

            System.out.println("The certificate is not valid until " +
                certificate.getNotBefore());
            throw e;
        }
/*
Step 11
*/
        ISamlResponseValidator validator = factory.getResponseValidator();
/*
Step 12
*/
        SamlResponse response = validator.processSamlResponseXml(ssoToken);
        System.out.println("The SSO token is valid.");
        System.out.println(response);
/*
Step 13
*/
        displaySamlResponse(response);
    }
/*
Step 14
*/
    private static CentrifySamlFactory getCentrifySamlFactory(
            CentrifySaasService service, String appKey, String spAudience) throws Exception
{
/*
Step 15
*/
        System.out.println("Getting metadata");
        // The CentrifySaasService will login automatically
```

```
        String metadata = service.getSAMLMetadataForApp(appKey);
        System.out.println("Successfully got metadata");

        // if the appKey doesn't match any existing application. The result is empty.
        if(metadata.isEmpty()) {
            throw new Exception("The metadata for appKey \"" + appKey + "\" is empty.");
        }

        String spIssuerName = "not used";
/*
Step 16
*/
        return new CentrifySamlFactory(metadata, spIssuerName, spAudience);
    }
/*
Step 17
*/
    private static void displaySamlResponse(SamlResponse response) {
        System.out.println();
/*
Step 18
*/
        System.out.println("The id is \"" + response.getNameid() + "\"");

        // the attribute name is case sensitive
        SamlResponse.Attribute emailAttr = response.getAttribute("Email");
        if (emailAttr != null) {
/*
Step 19
*/
            System.out.println(" Display Email attributes");
            System.out.println(" name format = " + emailAttr.getNameFormat());
            System.out.println(" Display attributes with \"format\"");
            List<SamlResponse.AttributeValue> attrValues = emailAttr.getValues();
            for (SamlResponse.AttributeValue attrValue : attrValues) {
                System.out.println("  " + attrValue.getFormat() + ": "
                        + attrValue.getValue());
            }
/*
Step 20
*/
            System.out.println(" Display attributes without \"format\"");
            List<String> attrStringValues = emailAttr.getStringValues();
            System.out.println("  " + attrStringValues);
        } else {
            System.out.println(" no Email attribute");
        }
/*
Step 21
*/
        List<String> groupValues = response.getAttributeValues("Groups");
        if (groupValues != null) {
            System.out.println(" Display Groups attributes");
            System.out.println(" Groups =  " + groupValues);
        } else {
            System.out.println(" no Groups attribute");
        }
    }
}
```

SamlResponseValidatorExample1

This sample hows how to create a `SamlResponseValidator` object and use it to validate a SAML response.

The sample code file `SamlResponseValidatorExample1.java` performs the following actions:

1. Imports Java libraries. Step 1
2. Imports an Apache security library. Step 2
3. Imports Centrify libraries. Step 3
4. Defines the `main` routine, which reads a base64-encoded SAML response from a file, validates the response, and displays the parsed result. Step 4
5. Calls the `getCertificate` subroutine (Step 15) to load the X509 certificate from a file. This certificate is used to validate the SAML response. Step 5
6. Instantiates a `SamlResponseValidator` object (see "SamlResponseValidator" on page 176). Step 6
7. Calls the `getBase64SamlResponse` subroutine (Step 16) to load the Base64-encoded SAML response from a file. This file could have been sent to your web service from your mobile application, for example. Step 7
8. Validates and parses the SAML response. If the `processSamlResponse` method does not throw an exception, the validation was successful. Step 8

   Note If the SAML response is in XML format, call the `processSamlResponseXML` method instead

9. Calls the `displaySamlResponse` subroutine (Step 10) to display the contents of the SAML response. Step 9
10. Defines a subroutine to display the data returned in the SAML response. Step 10
11. Uses methods of the `SamlResponse` class (see "SamlResponse" on page 162) to get the name identifier and the Active Directory Email attribute from the SAML response. Step 11
12. Uses methods of the `SamlResponse.Attribute` subclass (see "SamlResponse.Attribute" on page 168) to get the format and values of the Email attribute. Step 12
13. Illustrates the use of the `Attribute.getStringValues` method (page 171) to get the values of the Email attribute without the format. Step 13.
14. Illustrates the use of the `SamlResponse.GetAttributeValues` method (page 165) to get the values of an attribute. Step 14

Note If you only need to get the attribute values, you can call the `response.getAttributeString` method instead.

15 Defines a subroutine that uses the Java `CertificateFactory` class to load the signing certificate from a file. Step 15

16 Defines a subroutine that loads a SAML response from a file into a string. Step 16

Note The path to the SAML response file is provided in the call to the `main` routine (Step 4).

```
package com.centrify;
/*
Step 1
*/
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;
import java.security.PublicKey;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.List;
/*
Step 2
*/
import org.apache.xml.security.keys.content.x509.XMLX509Certificate;
/*
Step 3
*/
import com.centrify.cloud.saas.ISamlResponseValidator;
import com.centrify.cloud.saas.SamlResponse;
import com.centrify.cloud.saas.SamlResponseValidator;

public class SamlResponseValidatorExample1 {
/*
Step 4
*/
    public static void main(String[] args) throws Exception {
        if (args == null || args.length != 1) {
            System.out.println("Usage: <file path to base64 SAML response>");
            return;
        }
        // The public key that matches the private key used to sign the SAML Response
/*
Step 5
*/
        PublicKey idpPublicKey = getCertificate().getPublicKey();

        // Case-sensitive. This value must match the Issuer field of the
        // Application Settings tab for the application in Centrify Cloud Manager.
        String idpIssuerName = "Centrify";

        // Case-sensitive. The URI of the audience to which the response is addressed.
        String audience = "https://sp.centrify.com/login";
/*
Step 6
*/
        // Create the SamlResponseValidator
        ISamlResponseValidator validator = new SamlResponseValidator(
                idpPublicKey, idpIssuerName, audience);
/*
Step 7
```

```
*/
        String base64SamlResponse = getBase64SamlResponse(args[0]);
/*
Step 8
*/
        SamlResponse response = validator.processSamlResponse(base64SamlResponse);

        System.out.println("The SAML response is valid.");
        System.out.println(response);
/*
Step 9
*/
        displaySamlResponse(response);

    }
/*
Step 10
*/
    private static void displaySamlResponse(SamlResponse response) {
        System.out.println();
/*
Step 11
*/
        System.out.println("The id is \"" + response.getNameid() + "\"");

        // the attribute name is case sensitive
        SamlResponse.Attribute emailAttr = response.getAttribute("Email");
/*
Step 12
*/
        if (emailAttr != null) {
            System.out.println(" Display Email attributes");
            System.out.println(" name format = " + emailAttr.getNameFormat());
            System.out.println(" Display attributes with \"format\"");
            List<SamlResponse.AttributeValue> attrValues = emailAttr.getValues();
            for (SamlResponse.AttributeValue attrValue : attrValues) {
                System.out.println("  " + attrValue.getFormat() + ": "
                        + attrValue.getValue());
            }
/*
Step 13
*/
            System.out.println(" Display attributes without \"format\"");
            // if don't care the format,
            // you can call Attribute.getStringValues() instead
            List<String> attrStringValues = emailAttr.getStringValues();
            System.out.println("  " + attrStringValues);
        } else {
            System.out.println(" no Email attribute");
        }
/*
Step 14
*/
        List<String> groupValues = response.getAttributeValues("Groups");
        if (groupValues != null) {
            System.out.println(" Display Groups attributes");
            System.out.println(" Groups =  " + groupValues);
        } else {
            System.out.println(" no Groups attribute");
        }
    }
/*
Step 15
*/
    private static X509Certificate getCertificate() throws Exception {
```

```
            CertificateFactory certFact = CertificateFactory
                    .getInstance(XMLX509Certificate.JCA_CERT_ID);

            FileInputStream fis = new FileInputStream("sample.cer");

            try {
                return (X509Certificate) certFact.generateCertificate(fis);
            } finally {
                fis.close();
            }
        }
    /*
    Step 16
    */
        private static String getBase64SamlResponse(String file) throws IOException {
            StringBuilder sb = new StringBuilder();
            FileReader reader = new FileReader(file);
            try {
                int c;
                while ((c = reader.read()) != -1) {
                    sb.append((char) c);
                }
            } finally {
                reader.close();
            }
            return sb.toString();
        }

    }
```

SAML J2EE Reference

This chapter describes the classes, methods, and properties in the Centrify SAML J2EE API. The classes for implementing high-level SAML operations are defined in the `com.centrify.cloud.saas` package and consist of:

Class	Description
`AbstractLoginServlet`	An abstract class that you must override to implement a servlet module that handles login requests and validates SAML responses.
`AbstractSamlFilter`	An abstract class that you must override to implement a filter that you use to protect your internal resources from unauthorized users.
`AbstractServletConfigWrapper`	An abstract class that specifies a consistent interface to be used between a servlet container and the login servlet or filter.
`FilterConfigWrapper`	Specifies the interface to be used by the filter to get configuration information from the servlet container during initialization.
`SamlPrincipal`	A token that indicates that the user is authenticated.
`ServletConfigWrapper`	Specifies the interface to be used by the login servlet to get configuration information from the servlet container during initialization.
`ServletContextConfigWrapper`	Specifies the interface to be used by the login servlet to get context-wide configuration information during initialization.

Class: AbstractLoginServlet

An abstract class that you must override to implement a servlet module that handles login requests and validates SAML responses.

Syntax

```
abstract class AbstractLoginServlet
```

Field

The `AbstractLoginServlet` class provides the following field:

Field	Description
`Log`	The servlet log file.

Methods

The `AbstractLoginServlet` class provides the following methods:

Method	Description
`doPost`	Allows the servlet to support HTTP POST requests.
`getDefaultHomepage`	Returns the default home page URL for the login servlet to use for redirects.
`getValidator`	Returns a processor that generates SAML response validators.
`doDelete`	Inherited from `javax.servlet.http.HttpServlet`.
`doGet`	Inherited from `javax.servlet.http.HttpServlet`.
`doHead`	Inherited from `javax.servlet.http.HttpServlet`.
`doOptions`	Inherited from `javax.servlet.http.HttpServlet`.
`doPut`	Inherited from `javax.servlet.http.HttpServlet`.
`doTrace`	Inherited from `javax.servlet.http.HttpServlet`.
`getLastModified`	Inherited from `javax.servlet.http.HttpServlet`.
`service`	Inherited from `javax.servlet.http.HttpServlet`.
`destroy`	Inherited from `javax.servlet.GenericServlet`.
`getInitParameter`	Inherited from `javax.servlet.GenericServlet`.
`getInitParameterNames`	Inherited from `javax.servlet.GenericServlet`.
`getServletConfig`	Inherited from `javax.servlet.GenericServlet`.
`getServletContext`	Inherited from `javax.servlet.GenericServlet`.
`getServletInfo`	Inherited from `javax.servlet.GenericServlet`.
`getServletName`	Inherited from `javax.servlet.GenericServlet`.
`init`	Inherited from `javax.servlet.GenericServlet`.
`log`	Inherited from `javax.servlet.GenericServlet`.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

You override this abstract class to create a login servlet for your application. The login servlet must be able to receive and validate SAML responses so that you can log in users who have been authenticated by Centrify Cloud Service.

See Also

SamlRequestGenerator

SamlResponseValidator

Constructor: AbstractLoginServlet

Creates a new `AbstractLoginServlet` object.

Syntax

```
public AbstractLoginServlet()
```

Field: Log

The servlet log file.

Class

`AbstractLoginServlet`

Syntax

```
final org.slf4j.Logger log
```

Method: doPost

Allows the servlet to support HTTP POST requests.

Class

`AbstractLoginServlet`

Syntax

```
void doPost(
      HTTPServletRequest hreq
      HTTPServletResponse hresp
)
```

Parameters

`hreq` The HTTP request, containing the request received from the client.

`hresp` An HTTP response body to be sent to the client.

Errors

The `doPost` method may throw one of the following exceptions:

`IOException` An I/O operation failed or was interrupted.

`ServletException` The method encountered an error. This exception can pass on the exception that originally caused the error.

Discussion

This method overrides the `javax.servlet.http.HttpServlet.doPost` method so that the login servlet can handle HTTP `POST` requests. The login servlet tests a `POST` request to see if it includes a `SamlPrincipal` token. If it does, the servlet redirects the request to the application to log in the user. If the POST request does not include a `SamlPrincipal` token, the servlet sends an authentication request to the identity provider.

You must override the `getDefaultHomepage` and `getValidator` methods to enable this facility.

See Also

SamlPrincipal

getDefaultHomepage

getValidator

SamlRequestGenerator

SamlResponseValidator

Method: getDefaultHomepage

Returns the default home page URL for the login servlet to use for redirects.

Class

`AbstractLoginServlet`

Syntax

```
abstract String getDefaultHomepage()
```

Return value

The default home page URL.

Discussion

When your login servlet receives an HTTP request that does not contain a SAML response, it has to redirect the message back to the sending client. If the HTTP message is lacking a redirect URL, you can use this method to obtain the default URL to which you can redirect the request.

See Also

getIdpHttpPostUrl

Method: getValidator

Returns a processor that generates SAML response validators.

Class

`AbstractLoginServlet`

Syntax

```
abstract ISamlResponseValidator getValidator()
```

Return value

A processor you can use to generate a SAML authentication response validator.

Discussion

When you receive a SAML authentication response, you need to make sure the signature is valid and parse the response so you can extract any user attributes that you need in order to log in the user. You use a `SamlResponseValidator` object to perform this validation. You implement the `getValidator` method to return a validator object to the servlet container.

See Also

SamlResponseValidator

CentrifySamlFactory.getResponseValidator

Class: AbstractSamlFilter

An abstract class that you must override to implement a filter that you use to protect your internal resources from unauthorized users.

Syntax

```
abstract class AbstractSamlFilter
```

Field

The `AbstractSamlFilter` class provides the following field:

Field	Description
`responseHtmlFormTemplate`	A template for an HTML form.

Methods

The `AbstractSamlFilter` class provides the following methods:

Method	Description
`destroy`	Called by the servlet container when the filter is being taken out of service.
`doFilter`	Filters login requests.
`generateForm`	Generates the HTML form needed to send a SAML authentication request to the identity provider.
`getIdpHttpPostUrl`	Returns the sign-in URL for Centrify Cloud Service.
`getSamlRequestGenerator`	Returns an `ISamlRequestGenerator` object, which generates SAML authentication requests.
`init`	Sends initialization configuration information to the filter.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Discussion

This class implements the class `javax.servlet.Filter`. When this filter receives a login request, it determines whether the client is authenticated. If not, the filter sends a SAML `AuthnRequest` message to the identity provider through the browser that sent the request.

Your override of this filter must implement the `getSamlRequestGenerator` and `getIdpHttpPostUrl` methods. Override of the other methods is optional.

See Also

SamlRequestGenerator

SamlResponseValidator

Constructor: AbstractSamlFilter

Creates a new `AbstractSamlFilter` object.

Syntax

```
public AbstractSamlFilter()
```

Field: responseHtmlFormTemplate

A template for an HTML form.

Class

`AbstractSamlFilter`

Syntax

```
String responseHtmlFormTemplate
```

Example

The following code sample illustrates the use of the `responseHtmlFormTemplate` field:

```
/**
protected String generateForm(String samlRequest, String relayState) {
    return String.format(responseHtmlFormTemplate,
        getIdpHttpPostUrl()
        , samlRequest
        , relayState
     );
}
```

Method: destroy

Called by the servlet container when the filter is being taken out of service.

Class

`AbstractSamlFilter`

Syntax

```
void destroy()
```

Discussion

Use this method to clean up any resources or perform any other duties necessary when the filter is taken out of service.

Method: doFilter

Filters login requests.

Class

`AbstractSamlFilter`

Syntax

```
void doFilter(
      ServletRequest request
```

```
      ServletResponse response
      FilterChain chain
)
```

Parameters

request The login request received from the client.

response An object that the servlet container can send to the filter to help it send a response to the client.

chain An object that the servlet container can send to the filter to give a view into the invocation chain of the request.

Errors

The `doFilter` method may throw one of the following exceptions:

`IOException` An I/O operation failed or was interrupted.

`ServletException` The method encountered an error. This exception can pass on the exception that originally caused the error.

Discussion

The `doFilter` method is an implementation of the `doFilter` method in interface `javax.servlet.Filter`. When the filter receives a request to access protected resources, this method determines whether the client sending the request is authenticated. If not, it returns a SAML `AuthnRequest` message.

See Also

getRequestGenerator

Method: generateForm

Generates the HTML form needed to send a SAML authentication request to the identity provider.

Class

`AbstractSamlFilter`

Syntax

```
String generateForm(
      String samlRequest
      String relayState
)
```

Parameters

samlRequest The SAML `AuthnRequest` to be added to the HTML form.

relayState The SAML `RelayState` token to be passed to the identity provider.

Return value

The authentication request in an HTML form ready to be sent back to the client for forwarding to the identity provider.

Discussion

Because the Centrify cloud server uses HTTP POST binding, you should send the SAML `AuthnRequest` message as Base64-encoded XML to Centrify cloud. You can use the `SamlRequestGenerator.generateRequest` method to generate a SAML `AuthnRequest` message.

See Also

SamlRequestGenerator.generateRequest

Method: getIdpHttpPostUrl

Returns the sign-in URL for Centrify Cloud Service.

Class

```
AbstractSamlFilter
```

Syntax

```
String getIdpHttpPostUrl()
```

Return value

The identity provider (IdP) URL; this is the string that appears in the Cloud Manager window for your application in the **Identity Provider Sign-in URL** field.

Discussion

When the login sequence is initiated by the user attempting to log in directly to your SaaS application from their computer, this filter provides single-sign-on by redirecting the request to Centrify cloud. To do so, implement this method, which provides the redirection URL.

See Also

getRequestGenerator

SamlRequestGenerator

getDefaultHomepage

Method: getSamlRequestGenerator

Returns an `ISamlRequestGenerator` object, which generates SAML authentication requests.

Class

`AbstractSamlFilter`

Syntax

```
ISamlRequestGenerator getSamlRequestGenerator()
```

Return value

An interface for an object that generates a SAML authentication request.

Discussion

When the login sequence is initiated by the user attempting to log in directly to your SaaS application from their computer, this filter provides single-sign-on by redirecting the request to Centrify cloud. To enable this capability, you must implement this method so that the filter can generate a SAML authentication request (`AuthnRequest`) addressed to the identity provider.

See Also

getRequestGenerator

SamlRequestGenerator

getDefaultHomepage

Method: init

Sends initialization configuration information to the filter.

Class

`AbstractSamlFilter`

Syntax

```
void init(
      FilterConfig arg0
)
```

Parameters

`arg0` Configuration parameters for the filter.

Errors

The `init` method may throw the following exception:

`ServletException` The method encountered an error. This exception can pass on the exception that originally caused the error.

Discussion

This method is an implementation of the `init` method in interface `javax.servlet.Filter`.

The servlet container calls the `init` method when the filter is being put into service to pass configuration information to the filter.

See Also

FilterConfigWrapper

getInitParameter

Class: AbstractServletConfigWrapper

An abstract class that specifies a consistent interface to be used between a servlet container and the login servlet or filter.

Syntax

```
abstract class AbstractServletConfigWrapper
extends ValidationException
```

Implemented by

`FilterConfigWrapper`
`ServletConfigWrapper`
`ServletContextConfigWrapper`

Methods

The `AbstractServletConfigWrapper` class provides the following methods:

Method	Description
`getBoolean`	Returns a Boolean value corresponding to a specific key.
`getInitParameter`	Abstract method that returns a string containing the value corresponding to a specific key.
`getInteger`	Returns an integer value corresponding to a specific key.
`getString`	Returns a string value corresponding to a specific key.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.

Method	Description
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Constructor: AbstractServletConfigWrapper

Creates a new `AbstractServletConfigWrapper` object.

Syntax

```
public AbstractServletConfigWrapper().
```

Method: getBoolean

Returns a Boolean value corresponding to a specific key.

Class

```
AbstractServletConfigWrapper
```

Syntax

```
boolean getBoolean(
      String key,
)
```

```
boolean getBoolean(
      String key,
      boolean defaultValue,
)
```

Parameters

`key` The name of the Boolean configuration parameter you want to retrieve.

`defaultValue` The default value of the parameter.

Return value

The value of the specified configuration parameter. Returns an error if the parameter does not exist and no default value is specified. Returns the default value the parameter does not exist and a default is specified.

Errors

The `getBoolean(String)` method may throw the following exception:

`ServletException` The method encountered an error. This exception can pass on the exception that originally caused the error.

Discussion

The `getBoolean(String)` method returns `true` only if the value of the specified parameter is `"true"` (ignoring case). The `getBoolean(String,boolean)` method returns true only if the value of the specified parameter is `"true"` or if the parameter does not exist and the default value is `"true"`.

See Also

getInteger

getString

Method: getInitParameter

Abstract method that returns a string containing the value corresponding to a specific key.

Class

`AbstractServletConfigWrapper`

Syntax

```
abstract String InitParameter(
      String key,
)
```

Parameters

`key` The name of the configuration parameter you want to retrieve.

Return value

A string containing the value of the specified configuration parameter. Returns `null` if the parameter does not exist.

Discussion

This abstract method must be implemented for each type of configuration file that is to be read. The `getBoolean`, `getInteger`, and `getString` methods all call the `getInitParameter` method to obtain the value of an initialization parameter.

See Also

getBoolean

getInteger

getString

FilterConfigWrapper.getInitParameter

ServletConfigWrapper.getInitParameter

ServletContextConfigWrapper.getInitParameter

Method: getInteger

Returns an integer value corresponding to a specific key.

Class

`AbstractServletConfigWrapper`

Syntax

```
boolean getInteger(
      String key,
)
```

```
boolean getInteger(
      String key,
      int defaultValue,
)
```

Parameters

`key` The name of the integer configuration parameter you want to retrieve.

`defaultValue` The default value of the parameter.

Return value

The decimal integer value of the specified configuration parameter. Returns `null` if the parameter does not exist and no default value is specified. Returns the default value if the parameter does not exist and a default is specified.

Errors

The `getInteger` method may throw one of the following exceptions:

`ServletException` The method encountered an error. This exception can pass on the exception that originally caused the error.

`NumberFormatException` The value is not in integer format.

Discussion

The `getInteger` method returns a value only if all the characters in the string are decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value. Otherwise, it returns an error.

See Also

getBoolean

getString

Method: getString

Returns a string value corresponding to a specific key.

Class

`AbstractServletConfigWrapper`

Syntax

```
boolean getInteger(
      String key,
)
```

```
boolean getInteger(
      String key,
      int defaultValue,
)
```

Parameters

`key` The name of the integer configuration parameter you want to retrieve.

`defaultValue` The default value of the parameter.

Return value

The decimal integer value of the specified configuration parameter. Returns `null` if the parameter does not exist and no default value is specified. Returns the default value if the parameter does not exist and a default is specified.

Errors

The `getInteger` method may throw one of the following exceptions:

`ServletException` The method encountered an error. This exception can pass on the exception that originally caused the error.

`NumberFormatException` The value is not in integer format.

Discussion

The `getInteger` method returns a value only if all the characters in the string are decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value. Otherwise, it returns an error.

See Also

getBoolean

getInteger

Class: FilterConfigWrapper

Specifies the interface to be used by the filter to get configuration information from the servlet container during initialization.

Syntax

```
class FilterConfigWrapper
```

Implements

`AbstractServletConfigWrapper`

Methods

The `FilterConfigWrapper` class provides the following methods:

Method	Description
`getInitParameter`	Returns a string containing the value corresponding to a specific key.
`getBoolean`	Returns a Boolean value corresponding to a specific key. Inherited from `AbstractServletConfigWrapper`.
`getInteger`	Returns an integer value corresponding to a specific key. Inherited from `AbstractServletConfigWrapper`.
`getString`	Returns a string value corresponding to a specific key. Inherited from `AbstractServletConfigWrapper`.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Constructor: FilterConfigWrapper

Creates a new `FilterConfigWrapper` object.

Syntax

```
public FilterConfigWrapper(FilterConfig config).
```

Parameters

`config` Configuration parameters for the filter.

Method: getInitParameter

Returns a string containing the value corresponding to a specific key.

Class

`FilterConfigWrapper`

Syntax

```
abstract String InitParameter(
      String key,
)
```

Parameters

`key` The name of the configuration parameter you want to retrieve.

Return value

A string containing the value of the specified configuration parameter. Returns `null` if the parameter does not exist.

Discussion

This method calls the `javax.servlet.FilterConfig.getInitParameter` method.

See Also

AbstractServletConfigWrapper.getInitParameter

ServletConfigWrapper.getInitParameter

ServletContextConfigWrapper.getInitParameter

Class: SamlPrincipal

A token that indicates that the user is authenticated.

Syntax

```
class SamlPrincipal
extends java.lang.Object
implements java.security.Principal
```

Methods

The `SamlPrincipal` class provides the following methods:

Method	Description
`getName`	Returns the name of the authorized client represented by the `Principal` object.
`getPrincipal`	Extracts the `SamlPrincipal` token from an `HttpServletRequest` object.
`getResponse`	Returns the SAML response from the `SamlPrincipal` object.
`removePrincipal`	Removes the `SAMLPrincipal` token from an `HttpServletRequest` object.
`setPrincipal`	Adds a SAML `Principal` token to an `HttpServletRequest` object.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`toString`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.security.principal`.
`hashCode`	Inherited from `java.security.principal`.
`toString`	Inherited from `java.security.principal`.

Discussion

You add a `SamlPrincipal` object as an attribute to an `HttpServletRequest` object to indicate that a user is authenticated; that is, that you received a SAML response indicating the user is authenticated and you validated that message.

Note The `SamlPrincipal` object uses the HTTP session attribute key `"saml-principal"`. Do not use that string for any other purpose.

Constructor: SamlPrincipal

Creates a new `SamlPrincipal` object.

Syntax

```
public SamlPrincipal(
```

```
      SamlResponse response
)
```

Parameters

response A validated and parsed SAML response.

Method: getName

Returns the name of the authorized client represented by the `Principal` object.

Class

`SamlPrincipal`

Syntax

```
String getName()
```

Return value

The SAML name identifier used to identify a user in the SAML response.

Example

The following code sample illustrates the use of the `SamlPrincipal.getName` method:

```
/**
SamlPrincipal principal = SamlPrincipal.getPrincipal(request);
if(principal != null){
    out.append(principal.getName());
} else{
    out.append("Guest");
}
```

Method: getPrincipal

Extracts the `SamlPrincipal` token from an `HttpServletRequest` object.

Class

`SamlPrincipal`

Syntax

```
SamlPrincipal getPrincipal(
      HTTPServletRequest hreq
)
```

Parameters

hreq The HTTP request, containing the request received from the client.

Return value

The `SamlPrincipal` object from the request object; or `null` if the request contains no Principal token.

Example

The following code sample illustrates the use of the `SamlPrincipal.getPrincipal` method:

```
/**
SamlPrincipal principal = SamlPrincipal.getPrincipal(request);
if(principal != null){
    out.append(principal.getName());
} else{
    out.append("Guest");
}
```

See Also

ISamlRequestGenerator

Method: getResponse

Returns the SAML response from the `SamlPrincipal` object.

Class

`SamlPrincipal`

Syntax

```
SamlResponse getResponse()
```

Return value

The validated SAML response that was used to instantiate the `SamlPrincipal` object.

Discussion

The `SamlResponse` object is a validated and parsed SAML response.

Example

The following code sample illustrates the use of the `SamlPrincipal.getResponse` method in a Java Server Pages (`*.jsp`) file:

```
    User Principal = <%= userPrincipal %> <br>
    Is User In Role <i><%=roleParam%></i> = <%=isUserInRole%><br>

<% if(principal instanceof SamlPrincipal) {
    SamlResponse samlResponse = ((SamlPrincipal)principal).getResponse();
%>
<h3>Saml Response:</h3>
    NameId = <%= samlResponse.getNameid() %><br>
    NameFormat = <%= samlResponse.getNameFormat() %><br>
    Attributes:<br>
    <% for(SamlResponse.Attribute attribute : samlResponse.getAttributes()) { %>
     Name = <b><%=attribute.getName() %></b><br>
```

```
     NameFormat = <%=attribute.getNameFormat() %><br>
     FriendlyName = <%=attribute.getFriendlyName() %><br>
        <% for(SamlResponse.AttributeValue attributeValue : attribute.getValues()) { %>
              Value = <%=attributeValue.getValue() %>
            &#40;<%=attributeValue.getFormat()%>&#41;<br>
        <% } %>
    <% } %>
<%
    }
%>
```

See Also

SamlResponse

Method: removePrincipal

Removes the SAMLPrincipal token from an HttpServletRequest object.

Class

SamlPrincipal

Syntax

```
void removePrincipal(
      HTTPServletRequest hreq
)
```

Parameters

hreq The HTTP request containing the SamlPrincipal token.

See Also

setPrincipal

Method: setPrincipal

Adds a SAML Principal token to an HttpServletRequest object.

Class

SamlPrincipal

Syntax

```
void setPrincipal(
      HTTPServletRequest hreq
)
```

Parameters

hreq The HTTP request to which you want to add the SamlPrincipal token.

Discussion

When you add a SAML Principal token to an `HttpServletRequest` object, you are certifying that the client who sent the HTTP request has been authenticated.

Example

The following code sample illustrates the use of the `SamlPrincipal.setPrincipal` method:

```
}
```

See Also

getPrincipal

removePrincipal

Class: ServletConfigWrapper

Specifies the interface to be used by the login servlet to get configuration information from the servlet container during initialization.

Syntax

```
class ServletConfigWrapper
```

Implements

`AbstractServletConfigWrapper`

Methods

The `ServletConfigWrapper` class provides the following methods:

Method	Description
`getInitParameter`	Returns a string containing the value corresponding to a specific key.
`getBoolean`	Returns a Boolean value corresponding to a specific key. Inherited from `AbstractServletConfigWrapper`.
`getInteger`	Returns an integer value corresponding to a specific key. Inherited from `AbstractServletConfigWrapper`.
`getString`	Returns a string value corresponding to a specific key. Inherited from `AbstractServletConfigWrapper`.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.

Method	Description
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Constructor: ServletConfigWrapper

Creates a new `ServletConfigWrapper` object.

Syntax

```
public ServletConfigWrapper().
```

Method: getInitParameter

Returns a string containing the value corresponding to a specific key.

Class

`ServletConfigWrapper`

Syntax

```
abstract String InitParameter(
      String key,
)
```

Parameters

`key` The name of the configuration parameter you want to retrieve.

Return value

A string containing the value of the specified configuration parameter. Returns `null` if the parameter does not exist.

Discussion

This method calls the `javax.servlet.ServletConfig.getInitParameter` method.

See Also

AbstractServletConfigWrapper.getInitParameter

FilterConfigWrapper.getInitParameter

ServletContextConfigWrapper.getInitParameter

Class: ServletContextConfigWrapper

Specifies the interface to be used by the login servlet to get context-wide configuration information during initialization.

Syntax

```
class ServletContextConfigWrapper
```

Implements

`AbstractServletConfigWrapper`

Methods

The `ServletContextConfigWrapper` class provides the following methods:

Method	Description
`getInitParameter`	Returns a string containing the value corresponding to a specific key.
`getBoolean`	Returns a Boolean value corresponding to a specific key. Inherited from `AbstractServletConfigWrapper`.
`getInteger`	Returns an integer value corresponding to a specific key. Inherited from `AbstractServletConfigWrapper`.
`getString`	Returns a string value corresponding to a specific key. Inherited from `AbstractServletConfigWrapper`.
`clone`	Inherited from `java.lang.Object`.
`equals`	Inherited from `java.lang.Object`.
`finalize`	Inherited from `java.lang.Object`.
`getClass`	Inherited from `java.lang.Object`.
`hashCode`	Inherited from `java.lang.Object`.
`notify`	Inherited from `java.lang.Object`.
`notifyAll`	Inherited from `java.lang.Object`.
`wait`	Inherited from `java.lang.Object`.

Constructor: ServletContextConfigWrapper

Creates a new `ServletContextConfigWrapper` object.

Syntax

```
public ServletContextConfigWrapper().
```

Method: getInitParameter

Returns a string containing the value corresponding to a specific key.

Class

`ServletContextConfigWrapper`

Syntax

```
abstract String InitParameter(
      String key,
)
```

Parameters

`key` The name of the configuration parameter you want to retrieve.

Return value

A string containing the value of the specified configuration parameter. Returns `null` if the parameter does not exist.

Discussion

This method calls the `javax.servlet.ServletContextConfig.getInitParameter` method.

See Also

AbstractServletConfigWrapper.getInitParameter

FilterConfigWrapper.getInitParameter

ServletConfigWrapper.getInitParameter

SAML J2EE Sample Code

As illustrated in Figure 1. Centrify cloud components, there are four main components involved in Centrify Cloud Service:

- A corporate or organization's network running the Centrify Cloud Proxy Server and Active Directory.
- Centrify cloud authentication service.
- One or more providers of Internet (that is, cloud) based software (software as a service, or SaaS) including web applications that may be on-premises or hosted remotely.
- Mobile or desktop devices that need to access the web applications, where the users of the devices belong to groups and roles set up in Active Directory.

Centrify Cloud Service provides authentication and single sign-on (SSO) services for the mobile devices and for the SaaS application.

This chapter walks through the J2EE code for sample web routines for a filter and a login servlet.

The filter checks login requests to see if the user has been authenticated. If not, it generates a SAML authentication request (`AuthnRequest`) addressed to Centrify Cloud Service. Centrify Cloud Service authenticates the user and sends a SAML response. If the user has been authenticated, the filter passes the login request to the SaaS application.

The login servlet processes and validates the SAML response.

SAMLFilter

This sample uses the `xxx` class (see "CentrifySaasService" on page 144) to ….

The sample code file `SAMLFilter.java` performs the following actions:

1. Imports J2EE libraries. Step 1
2. Imports Centrify SaaS libraries. Step 2
3. Implements the `SamlFilter` class by extending the `AbstractSamlFilter` class in the API (see "AbstractSamlFilter" on page 202). Step 3

   When a filter that is an instantiation of the `AbstractSamlFilter` class receives a login request, it determines whether the client is authenticated. If not, the filter sends a SAML `AuthnRequest` message to the identity provider through the browser that sent the request.

4 Implements the `getSamlRequestGenerator` method (see "getSamlRequestGenerator" on page 207). Step 4

   This method returns an `ISamlRequestGenerator` object created in the `init` method (see Step 10). The filter uses this object to generate the `AuthnRequest` message that it passes to the identity provider.

5 Implements the `getIdpHttpPostUrl` method (see "getIdpHttpPostUrl" on page 206). Step 5

   This method returns an `idpHttpPostUrl` object created in the `init` method (see Step 11). The filter uses this object to determine the address of the identity provider.

6 Implements the filter's `init` method. In the SDK sample code, the Tomcat server passes a configuration object containing the configuration parameters from the `web.xml` file to the filter. In this sample (sample1), the configuration parameters include the `idpPostUrl` and `spIssuerName` values. Step 6

7 Calls the `super.init` method to pass the configuration object to the `init` method of the parent class. Step 7

8 Instantiates a `filterConfigWrapper` object, passing it the configuration object (see "FilterConfigWrapper" on page 213). Step 8

9 Calls the `filterConfigWrapper.getString` method to get the value of the `spIssuerName` configuration parameter. Step 9

10 Instantiates an `ISamlRequestGenerator` object (see "SamlRequestGenerator" on page 161), using the value of the `spIssuerName` parameter. Step 10

   The filter container calls the `getSamlRequestGenerator` method implemented in Step 4 to obtain this object.

11 Calls the `filterConfigWrapper.getString` method to get the value of the `idpHttpPostUrl` configuration parameter. Step 11

   The filter container calls the `getIdpHttpPostUrl` method implemented in Step 5 to obtain this value.

```
package com.centrify.cloud.saas.example;
/*
Step 1
*/
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
/*
Step 2
*/
import com.centrify.cloud.saas.AbstractServletConfigWrapper;
import com.centrify.cloud.saas.ISamlRequestGenerator;
import com.centrify.cloud.saas.SamlRequestGenerator;
import com.centrify.cloud.saas.filter.AbstractSamlFilter;
import com.centrify.cloud.saas.filter.FilterConfigWrapper;
```

```
/*
Step 3
*/
public class SamlFilter extends AbstractSamlFilter {

    private static final String PARAM_ISSUER_NAME = "spIssuerName";

    private static final String PARAM_IDP_HTTP_POST_URL = "idpPostUrl";

    private ISamlRequestGenerator requestGenerator;
    private String idpHttpPostUrl;
/*
Step 4
*/
@Override
    protected ISamlRequestGenerator getSamlRequestGenerator() {
        return requestGenerator;
    }
/*
Step 5
*/
@Override
    protected String getIdpHttpPostUrl() {
        return idpHttpPostUrl;
    }
/*
Step 6
*/
@Override
    public void init(FilterConfig fConfig) throws ServletException {
/*
Step 7
*/
        super.init(fConfig);
/*
Step 8
*/
        AbstractServletConfigWrapper filterConfig = new FilterConfigWrapper(
              fConfig);
/*
Step 9
*/
        String spIssuerName = filterConfig.getString(PARAM_ISSUER_NAME);
/*
Step 10
*/
        requestGenerator = new SamlRequestGenerator(spIssuerName);
/*
Step 11
*/
        idpHttpPostUrl = filterConfig.getString(PARAM_IDP_HTTP_POST_URL);
    }

}
```

LoginServlet

This sample shows how to create a login servlet and use it to validate a SAML response.

The sample code file `LoginServlet.java` performs the following actions:

1 Imports Simple Logging Facade for Java libraries. Step 1

2 Imports Centrify SaaS libraries. Step 2

3 Implements the `LoginServlet` class by extending the `AbstractLoginServlet` class in the API (see "AbstractLoginServlet" on page 198). Step 3

When a filter that is an instantiation of the `AbstractSamlFilter` class receives a login request, it determines whether the client is authenticated. If not, the filter sends a SAML `AuthnRequest` message to the identity provider through the browser that sent the request.

4 Implements the servlet's `init` method. Step 4

5 Instantiates a `ServletConfigWrapper` object (see "ServletConfigWrapper" on page 219). Step 9

6 Instantiates a `SamlResponseValidator` object (see "SamlResponseValidator" on page 176). Step 6

The servlet container calls the `getValidator` method implemented in Step 9 to obtain this `SamlResponseValidator` object.

7 Uses the `ServletConfigWrapper` object to get the value of the default home page. Step 7

8 Logs a message if the default home page is not set. Step 8

9 Implements the `getValidator` method. Step 9

The `getValidator` method is called in Step 6.

10 Implements the `getDefaultHomepage` method, which returns the default home page set in the `init` method. Step 10

The `getDefaultHomepage` method is called by the servlet container to get the home page to which to send the login request.

```
package com.centrify.cloud.saas.example;
/*
Step 1
*/
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
/*
Step 2
*/
import com.centrify.cloud.saas.AbstractServletConfigWrapper;
import com.centrify.cloud.saas.ISamlResponseValidator;
import com.centrify.cloud.saas.servlet.AbstractLoginServlet;
import com.centrify.cloud.saas.servlet.ServletConfigWrapper;
/*
Step 3
*/
public class LoginServlet extends AbstractLoginServlet {

    private static final long serialVersionUID = 1L;

    private static final String PARAM_DEFAULT_HOME_PAGE = "defaultHomepage";
```

```
    final Logger log = LoggerFactory.getLogger(LoginServlet.class);

    ISamlResponseValidator samlValidator;
    String defaultHomepage;

/*
Step 4
*/
@Override
    public void init() throws ServletException {
/*
Step 5
*/
        AbstractServletConfigWrapper servletConfig = new ServletConfigWrapper(this);
/*
Step 6
*/
        samlValidator = CentrifySamlInitalizer.getFactory().getResponseValidator();
/*
Step 7
*/
        defaultHomepage = servletConfig.getString(
            PARAM_DEFAULT_HOME_PAGE, null);
/*
Step 8
*/
        if(defaultHomepage == null) {
            defaultHomepage = "/" + getServletContext().getServletContextName();
            log.debug("The parameter '" + PARAM_DEFAULT_HOME_PAGE + "' is not set.
                The default will be '" + defaultHomepage + "'");
        }
    }
/*
Step 9
*/
@Override
    protected ISamlResponseValidator getValidator() {
        return samlValidator;
    }
/*
Step 10
*/
@Override
    protected String getDefaultHomepage() {
        return defaultHomepage;
    }

}
```

Creating a SAML application profile

If an application that uses Centrify Cloud Service to implement single-sign-on (SSO) requests a security token (that is, a SAML response) in order to authenticate a user, the Centrify Cloud Service must have a SAML application profile in place for the token to work. This chapter documents how to configure the Generic SAML template in Centrify Cloud Manager to create a SAML application profile. See "SAML authentication involves receiving and validating a token" on page 24 for a general introduction to SAML.

If you are an administrator adding a SAML-enabled SaaS application to MyCentrify, or if you are adding a mobile application to the Centrify application and that mobile application has a SAML-enabled web service as a back-end, then you must configure the Generic SAML template for that application. The **Advanced** tab of that template requires a script, written in Javascript, that specifies the contents of the SAML response. Centrify Cloud Service executes the SAML script when generating the security token for the SAML-enabled application or web service. Without the SAML application profile and SAML script in the Generic SAML template, Centrify Cloud Service cannot generate the security token.

Note that this means that in order to install a mobile application in the Centrify application—if the mobile application has a SAML-enabled SaaS back-end—you must configure both the mobile application template (which creates a button for the application visible to users) and the Generic SAML template (which is not visible to users, but is necessary for the SAML interface).

Either the vendor of the SaaS application or the vendor of the mobile application that has a web-service back-end must provide configuration information to the administrator, including the SAML script for the **Advanced** tab of the template.

This chapter describes the SAML application profile, explains the steps needed to put SAML profiles in place for either a SaaS application or a mobile application with a web back-end, and documents the methods, variables, and functions needed in order to write a SAML script. The chapter includes these sections:

- Configure the SAML interface describes the information needed for a SAML application profile.
- Provide help for administrators shows the steps required to support an administrator who needs to configure a SAML application profile and provides a sample set of instructions that a software vendor can adapt to help an administrator install their product in Cloud Manager.
- Debugging your scripts provides some information that can help in finding problems in the SAML interface.

- Scripting environment reference documents the classes, methods, properties, and functions provided for use in SAML scripts.

To write a SAML script, you need to know how to write code in Javascript. You also need to know the basics of SAML authentication to understand how to specify a SAML assertion. This chapter provides some guidance about SAML configuration values, but for specifics you can consult the SAML specifications provided at http://saml.xml.org/saml-specifications. For an introduction to SAML, try the overviews provided at http://saml.xml.org/wiki/saml-introduction. In addition, Jeff Hodges, Neustar, *How to Study and Learn SAML*, 2006 (http://identitymeme.org/doc/draft-hodges-learning-saml-00.html) defines SAML terms and concepts and provides a reading list.

Configure the SAML interface

To configure a new SAML application profile for a SAML-enabled application or web service, an administrator must provide or obtain:

- A name for the application profile.
- An application ID, which is a name that is unique within the organization.
- A SAML script that defines the SAML assertion for the SAML-enabled application or web service.
- The URL at which the application or web service accepts SAML responses.
- The URL that identifies the Centrify Cloud Service as the SAML identity provider (IdP).

The administrator may customize the SAML application profile by specifying:

- The source for the user name to be included in the SAML assertion.

  The user name in the SAML assertion does not have to be the same as the user name that was authenticated to Centrify Cloud Service. Although Centrify Cloud Service requires the user's Active Directory login name for authentication, the user name presented to the web service can be something else. For example, the Threadbare.com web service might use a single user name to authenticate all of Acme Widget's users (`Employee@AcmeWidgets`). When John Doe of Acme Widgets uses the Threadbare mobile application for the first time, Centrify cloud requires him to provide his Active Directory login name (`John.Doe@AcmeWidgets.com`) for authentication purposes, but when Centrify Cloud Service prepares the SAML assertion, it substitutes `Employee@AcmeWidgets` for `John.Doe@AcmeWidgets.com`.

- A non-Centrify certificate and private key to be used to sign the SAML assertion or response.
- Other profile attributes (such as the icon and service description) that don't affect the application or SAML interface.

In order for the SAML-enabled application or web service to be able to read and respond to the SAML assertion, the administrator who is configuring the application profile in Centrify

Cloud Manager must communicate with the organization that is providing the SAML-enabled application or web service to provide:

- The security certificate used to sign the SAML assertion or response (the certificate contains the public key needed to verify the signature).
- The identity provider URLs (sign-on, sign-off, and error).

The administrator must get these attributes from the SAML application profile and provide them directly to the SAML service provider. Without this information, the SAML service provider cannot configure the SAML interface to work with the mobile application or Centrify Cloud Service.

How the administrator conveys these attributes to the developer of the SAML-enabled application or web service is completely up to the SAML service. Some service providers set up web pages that define these attributes and then accept the attributes through input fields; others require that the administrator contact the SAML service provider and give this information in person.

Provide help for administrators

Writing a SAML script is not a trivial task, especially for an administrator who may know nothing of SAML or Javascript. Therefore, the developer of the SAML interface for the application or web service should provide a SAML script and the other information necessary for configuring the SAML application profile for their application or web service. The SAML service provider can convey this information on request by email or personal contact, or provide it through a web interface, depending on the anticipated volume of requests and the preferences of the SAML service provider. In the case of a mobile application that uses a SAML-enabled web service as a back end, the vendor of the mobile application might find it in their best interest to act as an intermediary between the SAML provider and the administrator responsible for installing the application and configuring the application profile for the customer.

The steps for providing help are:

1 Determine the SAML requirements.

  The developer of the SAML interface for the application or web service must know:

  - What elements the SAML service requires in the SAML assertions it receives.

    For example, the SAML service might require the login user's email address (`emailAttr`).

  - What information an organization must supply directly to the SAML service, such as the certificate needed to verify the signature and a set of identity-provider URLs.

    For example, if the developer of the SAML interface is using the Centrify Java SAML SaaS API, they might require the SAML metadata downloaded from the application profile page in Cloud Manager (click the **Download Identity Provider SAML**

metadata button). (See getSAMLMetadataForApp and CentrifySamlFactory for more information about the SAML metadata and its use in the Centrify SAML SaaS API.)

- How to convey this information to the SAML service provider.

  For example, the SAML metadata can be sent as a file via a secure email.

2 Write a SAML script.

The rest of this chapter is addressed to the person responsible for writing the SAML script in the application profile. Although it's the administrator of the organization that has licensed the application who has to enter the script into the Cloud Manager application profile, it is the developer of the SAML interface who is best equipped to write the script.

Once you know what elements are required in a SAML assertion for your SAML service, you can write a SAML script to define those elements. "SAML authentication involves receiving and validating a token" on page 24 describes the steps that occur during SAML assertion creation and presentation so that you can understand the context in which the script runs. "The SAML script specifies the attributes in the token" on page 36 describes how to write the script. "Scripting environment reference" on page 233 lists and describes the Javascript objects and variables available for use in a SAML script.

3 Write administrator instructions.

When you have a SAML script ready and know the other requirements an administrator must meet to create a SAML application profile for your application or web service, you can write a set of instructions for the administrator to follow that include the SAML script to copy and paste into the generic SAML application profile. Sample SAML application profile instructions provides a sample set of instructions you can adapt.

4 Publish the administrator instructions.

Once you've created your administrator instructions, you must post them (including your SAML script) in a location where your customers can easily find and follow them, or provide some other mechanism for your customers to obtain them.

Sample SAML application profile instructions

Here is a set of generalized instructions for administrators who need to configure an application profile for a SAML-enabled SaaS application or web service. Directions for how to complete composing the instructions are enclosed in square brackets and indicated with italic text, [*`like this`*].

1 In Centrify Cloud Manager, on the **Apps** page, click **Add App**. In the Select Applications dialog, search for "Generic". Select the Generic SAML template and click **Add Apps**.

2 On the **Application Settings** tab of the Generic SAML template, click **Edit** in the upper right corner.

3 Enter an application name and description in the top two text boxes.

The application name need not be unique, but will be visible to users unless you deselect the **Show in Centrify** check box at the bottom of the template, as described in the next step.

4 If you are installing a SaaS application that is available for users to log in to directly, make sure the **Show in Centrify** checkbox is selected and upload an icon for the application by clicking the **Browse** button under **Upload New Image**. [*If you are providing the icon, give the name of the icon file and say where to get it.*] If you are installing a mobile application with a SaaS back-end, deselect the checkbox. In this case, you don't need an icon for the template, as it will not be visible to the user.

5 Enter a name or ID for the SaaS application or service in the **Application ID** text box.

The application ID must be unique among all the applications—whether visible in the Centrify application or not—installed by your organization in Cloud Manager.

6 Enter the following value in the **URL** text field: [*enter the URL at which your SAML application or service receives the SAML response*].

This is the address [to which the mobile application forwards *OR* at which the SaaS application receives] the SAML response.

7 Modify the URL in the **Issuer** text field by replacing `Generic SAML` with the contents of the **Application ID** text box.

8 [*Include this step only if you want to use your own certificate and private key.*] Place the [*insert name of the PKCS #12 (*.p12 or *.pfx) file containing your certificate and key*] file you received [*state how the administrator obtained or can obtain the file*] in a known location and click **Replace** under **Security Certificate**. Navigate to the file and click **Open**.

9 Select the user account mapping you want to use for authenticating the user.

Although Centrify Cloud Service requires the user's Active Directory login name for authentication, the user name presented to the web service can be something else. For example, the Threadbare.com web service might use a single user name to authenticate all of Acme Widget's users (`Employee@AcmeWidgets`). When John Doe of Acme Widgets uses the Threadbare mobile application for the first time, Centrify cloud requires him to provide his Active Directory login name (`John.Doe@AcmeWidgets.com`) for authentication purposes, but when Centrify Cloud Service prepares the SAML assertion, it substitutes `Employee@AcmeWidgets` for `John.Doe@AcmeWidgets.com`.

Under **Map to User Accounts**:

- To use an Active Directory attribute, select the **Use the following Active Directory field to supply the user name** radio button and enter the name of the AD field in the text box.

- To use the same user name for all users, select the **Everybody shares a single user name** radio button and enter the name in the text box.
- To use a custom script to map user names, click the **Use this Script** radio button and enter the script. Instructions for writing user-name-mapping scripts are included in the Cloud Manager online help.

10 Open the **Advanced** tab and click **Edit**. Paste the provided SAML script into the text box, replacing the template script that's there. [*Describe where to find the SAML script.*]

11 In order for the SAML interface to work correctly, you must transmit certain information to [*enter name of software vendor providing these instructions*].

Return to the **Application Settings** tab to get the following data:

- The application ID.
- [*Remove any of these you don't need.*] The contents of the **Identity Provider Sign-In URL**, **Identity Provider Error URL**, and **Identity Provider Sign-Out URL** text boxes.
- [*If you need a copy of the security certificate, include this bullet.*] The certificate needed to validate the signature of the SAML response. Click **Download** under **Security Certificate** for a copy of the certificate.
- [*If you want the metadata, include this bullet. Note that the metadata includes the certificate.*] The SAML metadata. Click **Download Identity Provider SAML Meta data** to download the metadata file.

12 Click **Save Changes**.

13 Send the information as follows: [*Insert instructions for transmitting information to the software developer responsible for the SAML interface*].

Debugging your scripts

There are some resources available on the Internet for debugging SAML interfaces. See, for example, http://www.coreblox.com/2011/06/collection-of-useful-saml-tools/ for a list of useful tools.

Scripting environment reference

Centrify Cloud Service creates a set of Javascript objects, global variables, and global functions for each SAML user session. These objects provide information that a custom SAML script can read and act on. Some of the objects also accept values that specify elements with the SAML assertion that Centrify Cloud Service presents to a SAML service.

This section describes the SAML user-session Javascript environment in which the user map script and the custom SAML script execute. The section describes each available object and its public properties and methods. It also describes available global variables and global functions.

"The SAML script specifies the attributes in the token" on page 36 describes how the Centrify Cloud Service works with a set of Javascript objects when executing the SAML script.

Class: LoginUser

Contains information about the user identity used to authenticate through the Centrify Cloud Service and the organization's directory service.

Properties

The `LoginUser` object's properties describe the user as he or she is presented to the SAML service. The `LoginUser` class provides the following properties:

Property name	Description
`UserName`	The user identity presented in the SAML assertion to the SAML service.
`GroupNames`	Groups in which the user is a direct member.
`EffectiveGroupNames`	Groups in which the user is an effective member.
`GroupDNs`	Distinguished names of groups in which the user is a direct member.
`EffectiveGroupDNs`	Distinguished names of groups in which the user is an effective member.

Methods

The `LoginUser` class has a single method:

Method	Description
`Get`	Returns an attribute from Active Directory.

Discussion

Centrify Cloud Service creates a single `LoginUser` object for each SAML user session; the object is an instance of the `LoginUser` class, and is a read/write object.

Property: UserName

The user identity presented in the SAML assertion to the SAML service.

Syntax

```
loginuser.UserName
```

Discussion

Centrify Cloud Service determines the user ID for this user session depending on the **Map to User Accounts** setting in the **Application Settings** tab. (These settings determine the user name, which is the user ID presented in the SAML assertion.) This property is synonymous with the global variable `LoginUsername`.

Property: GroupNames

Groups in which the user is a direct member.

Syntax

```
loginuser.GroupNames
```

Return value

An array of names of groups in which the user is a direct member.

Discussion

The groups come from the user's account in the organization's Active Directory.

Property: EffectiveGroupNames

Groups in which the user is an effective member.

Syntax

```
loginuser.EffectiveGroupNames
```

Return value

An array of names of groups in which the user is a direct member.

Discussion

The effective groups come from the user's account in the organization's Active Directory. An effective member of a group is either a direct member of the group or a member of a group that is in turn a direct or nested member of the group.

Property: GroupDNs

Distinguished names of groups in which the user is a direct member.

Syntax

```
loginuser.GroupDNs
```

Return value

An array of distinguished names of groups in which the user is a direct member.

Discussion

The groups come from the user's account in the organization's Active Directory.

Property: EffectiveGroupDNs

Distinguished names of groups in which the user is an effective member.

Syntax

```
loginuser.EffectiveGroupDNs
```

Return value

An array of distinguished names of groups in which the user is a direct member.

Discussion

The effective groups come from the user's account in the organization's Active Directory. An effective member of a group is either a direct member of the group or a member of a group that is in turn a direct or nested member of the group.

Method: Get

Returns an attribute from Active Directory.

Syntax

```
LoginUser.Get(ADKey)
```

Parameters

ADKey A string specifying the name of the attribute to retrieve.

Return value

A string value corresponding to the key specified in the parameter.

Class: Application

Contains the properties of the SAML service as they're defined in the SAML application profile.

The `Application` object is read-only. A script accesses the object's properties using the object's single public method.

Method Name	Description
`Get`	Returns a property value from the `Application` object.

Discussion

Centrify Cloud Service creates a single `Application` object for each SAML user session. The object is an instance of the `ReadOnlyDataEntity` class, and is a read-only object.

The `Application` object's properties describe the SAML service as it's defined in the SAML application profile. Create a SAML application profile in Cloud Manager using the Generic SAML application template (as described in "Configure the SAML interface" on page 229).

The following section describes the property arguments this method can take.

Method: Get

Returns a property value from the `Application` object.

Syntax

`Application.Get(`*`property`*`)`

Parameters

`property` The object property for which you want a value. Properties for which you can get values are shown in the following table. The property names are case sensitive.

Property Name	Description
_PartitionKey	The customer ID used to establish the user session. Example: BZ284.
_RowKey	The UUID (universally unique identifier) of the application.
Description	The text description of the web application entered in the description field of the **Application Settings** tab.
Icon	The graphic file used as the icon for this application as set by the **Upload New Image** interface on the **Application Settings** tab.
Issuer	The entity ID specified in the **Issuer** field of the **Application Settings** tab. This is typically a URL that identifies the Centrify Cloud Service as the SAML assertion's issuer. Synonymous with the global variable `Issuer`.
Name	The name of the application as entered in the **Application Settings** tab.
SamlScript	The custom SAML script set in the **Advanced** tab.
TemplateName	The type of generic application template used to define this web application's profile. Possible return values: • `Generic SAML` • `Generic User-Password`

Property Name	Description
Url	The contact URL specified in the **URL** field in the **Application Settings** tab. This is typically the URL to which the SAML assertion (in its enclosing SAML response) is sent. Synonymous with the global variable `ServiceUrl`.
UserName Strategy	The technique specified in the **Application Settings** tab to determine the user name (user identity) for a user session. Possible return values: • `ADAttribute`: Centrify Cloud Service sets the user name to the specified AD attribute of the current user. Centrify Cloud Service queries the AD proxy for the AD attribute. Centrify Cloud Service caches the user name so that it doesn't have to query the proxy for this user's future sessions. • `Fixed`: Centrify Cloud Service sets the user name to the value entered in the **Application Settings** tab. • `UseScript`: Centrify Cloud Service executes the user map script to determine the user name.
WebAppType	The authentication method used by the web application. Possible return values: • `SAML` • `UsernamePassword`

Discussion

Use the property values returned by this method to obtain values you wish to provide in the SAML response, then use the global functions described in "Global functions" on page 239 to add the values to the response. For example, to add the issuer of the SAML assertion to the SAML response, you could use:

```
setIssuer(Application.Get('Issuer'))
```

Global variables

Centrify Cloud Service creates a set of global variables for each SAML user session. These variables are synonyms for common attributes of the `LoginUser` and `Application` objects, and are a convenience: you can use a global variable instead of specifying a `LoginUser` attribute or using `Application.get` to read an `Application` attribute.

Global Variable	Description
ApplicationUrl	A read-only variable that contains the contact URL specified in the URL field in the **Application Settings** tab. Synonymous with the `Application` attribute `Url`.
Issuer	A read-only variable that contains the entity ID specified in the **Issuer** field of the **Application Settings** tab. Synonymous with the `Application` attribute `Issuer`.
LoginUsername	A read-write variable that contains the user identity (user name) presented in the SAML assertion to the SAML service. Centrify Cloud Service determines the user name for this user session depending on the **Map to User Accounts** setting in the **Application Settings** tab. Synonymous with the attribute `LoginUser.Username`.
ServiceUrl	A read-only variable that contains the contact URL specified in the **URL** field in the **Application Settings** tab. Synonymous with the `Application` attribute `Url`.

Global functions

Centrify Cloud Service provides a set of global functions available in a SAML user session that specify elements within a SAML assertion.

The private assertion object specifies how Centrify Cloud Service constructs the SAML assertion for a SAML user session. Assertion-set functions set the attributes of the private SAML assertion object. Most of these functions take as an argument the value for a specific SAML assertion element. The `setIssuer` function, for example, accepts an entity ID and uses it to specify the issuer URL in the SAML assertion.

Two of the assertion set functions, `setAttribute` and `setAttributeArray` specify a SAML response attribute by name and then specify a value for that attribute that is either a single argument or an array. Use these functions to add SAML assertion elements that can't be specified by any of the other assertion set functions.

The following table lists global assertion-set functions available in a user session.

global function	Description
setAttribute	Sets a specified SAML assertion element to a value.
setAttributeArray	Sets a specified SAML assertion element to an array.
setAudience	Specifies the audience in an audience restriction in the SAML assertion.
setAuthenticationMethod	Specifies the type of authentication used to authenticate the user.
setHttpDestination	Specifies the URL to which to post the SAML response.
setIssuer	Specifies the issuer in the SAML assertion.
setNameFormat	Specifies the format value in the SAML assertion's `NameID` element.
setRecipient	Specifies the recipient in the SAML assertion.
setRelayState	Specifies a `RelayState` parameter to send with the SAML response.
setServiceUrl	Specifies the resource to which the user is requesting access.
setSignatureType	Specifies what should be signed: the SAML assertion or the SAML response.
setSubjectConfirmationMethod	Specifies the SAML subject confirmation method.
setSubjectName	Specifies the user identity presented to the SAML service.
setVersion	Specifies the version of the SAML assertion.

Function: setAttribute

Sets a specified SAML assertion element to a value.

Syntax

`setAttribute(`*`elementName`*`, `*`elementValue`*`)`

Parameters

elementName A string that specifies the name of a SAML assertion element to set.
elementValue The value to set for the attribute.

Discussion

Use this function to set a single value for an attribute in the SAML response when there is no function specific to that attribute.

Example

```
setAttribute("Email", LoginUser.Get("mail"));
```

Specifies the SAML assertion element named `Email` to be set to the current user's email address in Active Directory.

Function: setAttributeArray

Sets a specified SAML assertion element to an array.

Syntax

```
setAttributeArray(elementName, elementArray)
```

Parameters

elementName A string that specifies the name of a SAML assertion element to set.
elementArray The array to set for the attribute.

Discussion

Use this function to set an array value for an attribute in the SAML response when there is no function specific to that attribute.

Example

```
setAttributeArray('Groups', LoginUser.GroupNames);
```

Specifies the SAML assertion element named `Groups` to be set to an array of names of groups in which the current user is a direct member.

Function: setAudience

Specifies the audience in an audience restriction in the SAML assertion.

Syntax

```
setAudience(audience)
```

Parameters

audience The audience to which you want to restrict the assertion. Typically a URL.

Example

```
setAudience("https://login/myapp.com");
```

Function: setAuthenticationMethod

Specifies the type of authentication used to authenticate the user.

Syntax

```
setAuthenticationMethod(authenticationUri)
```

Parameters

authenticationUri A URI as described in section 2.4.3 of the SAML 2.0 core specification. The same specification lists possible URI values in section 7.1.

Example

```
setAuthenticationMethod("urn:oasis:names:tc:SAML:1.0:am:password");
```

Specifies that the user was authenticated with a password.

Function: setHttpDestination

Specifies the URL to which to post the SAML response.

Syntax

```
setHttpDestination(responseUrl)
```

Parameters

responseUrl The URL to which to post the response.

Description

This function specifies the URL to use in the response's HTTP POST binding (the value in the `action=` argument). Normally you can get this URL by retrieving the `Application` property `Url` or by using the property's synonymous variable `ServiceUrl`.

Example

```
setHttpDestination(Application.Get('Url'));
```

Function: setIssuer

Specifies the issuer in the SAML assertion.

Syntax

```
setIssuer(issuer)
```

Parameters

issuer The URL of the issuer of the SAML response.

Description

This function specifies the URL of the issuer to use in the SAML assertion. Normally you can get this URL by retrieving the Application property `Issuer` or by using the property's synonymous variable `Issuer`.

Example

```
setIssuer(Issuer);
```

Function: setNameFormat

Specifies the format value in the SAML assertion's `NameID` element.

Syntax

`setNameFormat(`*format*`)`

Parameters

format The format for the `NameID` element.

Description

This function specifies the format value (the value following `Format=`) in the SAML assertion's `NameID` element. This element is only used in a SAML 2.0 assertion.

Function: setRecipient

Specifies the recipient in the SAML assertion.

Syntax

`setRecipient(`*recipient*`)`

Parameters

recipient The recipient for the SAML assertion.

Description

A web browser or a mobile application with a SAML-enabled SaaS back-end requests the security token (SAML response) from Centrify Cloud Service and then forwards it to the SaaS application. This function sets the value in the SAML assertion's `SubjectConfirmationData` element that the mobile application or the web browser reads in order to get the ultimate destination of the assertion. The recipient is typically a URL such as `https://login/myapp.com`.

Function: setRelayState

Specifies a `RelayState` parameter to send with the SAML response.

Syntax

`setRelayState(`*`relayState`*`)`

Parameters

`relayState` The value of the `relayState` parameter, as specified by the service provider.

Description

The `RelayState` parameter is a mechanism for conveying state information needed by the service provider. You must set this attribute only if the service provider (that is, the SaaS vendor) requires it. In that case, you must get the value from the SaaS vendor, as discussed in "Provide help for administrators" on page 230.

The `RelayState` parameter is specified in bindings and profiles specifications for SAML (see http://saml.xml.org/saml-specifications).

Function: setServiceUrl

Specifies the resource to which the user is requesting access.

Syntax

`setServiceUrl(`*`targetUrl`*`)`

Parameters

`targetUrl` The URL of the resource to which the user is requesting access.

Description

This method sets the value of the `TARGET` attribute in the HTML `form` element of the document used to transmit the SAML response. The `TARGET` attribute specifies the resource (such as a web page, file, or service) to which the user of the application is requesting access. Typically, this is the same URL as the destination to which the SAML response is sent (see setHttpDestination), but they can be different if so specified by the service provider.

Function: setSignatureType

Specifies what should be signed: the SAML assertion or the SAML response.

Syntax

`setSignatureType(`*`signingPref`*`)`

Parameters

`signingPref` A value indicating what should be signed: `Response` or `Assertion`.

Description

Either the SAML assertion or the containing SAML response can be signed, as specified by this method. If this method is not included in the script, the default is `Response`.

Function: setSubjectConfirmationMethod

Specifies the SAML subject confirmation method.

Syntax

`setSubjectConfirmationMethod(`*`methodUri`*`)`

Parameters

`methodUri` A URI indicating what SAML confirmation method should be used.

Description

This function specifies the SAML subject confirmation method identifier for the SAML assertion's binding. The SAML subject confirmation method is described in bindings and profiles specifications for SAML (see http://saml.xml.org/saml-specifications).

Example

The following line specifies the Bearer confirmation method.

```
setSubjectConfirmationMethod(""urn:oasis:names:tc:SAML:2.0:cm:bearer">
<SubjectConfirmationData InResponseTo="_1234567890"
Recipient="https://www.serviceprovider.com/saml/consumer"
NotOnOrAfter="2004-03-19T13:27:00Z"
</SubjectConfirmationData")
```

Function: setSubjectName

Specifies the user identity presented to the SAML service.

Syntax

`setSubjectName(`*`username`*`)`

Parameters

`username` The user name presented to the service provider.

Description

This method specifies the subject in the SAML assertion, which is the user identity (that is, the user name) presented to the SAML service. You can get this name by retrieving the `LoginUser.Username` property or by using the property's synonymous variable `LoginUsername`.

Function: setVersion

Specifies the version of the SAML assertion.

Syntax

`setVersion(`*`samlVersion`*`)`

Parameters

`samlVersion` The SAML version number. Specify `1` for version 1.1 or `2` for version 2.0.

Description

The default SAML version number is 2.0 if this method isn't present in the script.